### containerd使用手册

**前言：**

1.蓝色字体表示命令行命令，正式执行时不要复制前面的#号，#号只是提示应该使用root权限操作

2.绿色字体表示注释，有时注释太多就不用绿色表示了

3.注意：本文档的所有操作请先在在测环境进行实践，请不要直接在真实的服务器中操作！

**版权声明**：

本文档以开源的形式发布，所有条款如下：

1. 无担保：作者不保证文档内容的准确无误，亦不承担由于使用此文档所导致的任何后果

2. 自由使用：任何人可以出于任何目的而自由地 阅读/链接/打印/转载/引用/再创作 此文档，无需任何附加条件

若您 阅读/链接/打印/转载/引用/再创作 本文档，则说明接受以上2个条款。

作者：李茂福

时间：2023-09-28

**第0章、安装containerd**

**★首先加载模块**

# cat > /etc/modules-load.d/containerd.conf <<EOF

overlay

br\_netfilter

EOF

# modprobe overlay

# modprobe br\_netfilter

**★安装containerd.io软件**

可以使用阿里的源或docker官方的软件源

# cd /etc/yum.repos.d/

# wget https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo #aliyun的源

或者： https://download.docker.com/linux/centos/docker-ce.repo #官方的源



然后在/etc/yum.repos.d/目录下多了一个名为docker-ce.repo的文件

如果下载不了，可以在此目录下自己创建一个docker-ce.repo文件，内容如下：

[docker-ce-stable]

name=Docker CE Stable - $basearch

baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/$basearch/stable

enabled=1

gpgcheck=1

gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

# yum install containerd.io -y

# containerd --version #查看版本

containerd containerd.io 1.6.18 2456e983eb9e37e47538f59ea18f2043c9a73640

**第1章、containerd初始设置**

**★配置containerd为systemd服务**

# mkdir -p /etc/containerd

# containerd config default > /etc/containerd/config.toml #生成containerd的配置文件

# vi /etc/containerd/config.toml #修改以下2行配置

SystemdCgroup = true

sandbox\_image = "cof-lee.com:5443/k8s/pause:3.9" #和k8s需要的pause镜像版本保持一致

#如果要启用CRI-Plugin，注释掉其中的 disabled\_plugins = ["cri"]

#再重启containerd即可有 unix:///run/containerd/containerd.sock 接口

# systemctl enable containerd

# systemctl start containerd

**★信任私有镜像仓库**

# vi /etc/containerd/config.toml #添加或修改以下几行配置

**#信任私有镜像仓库ssl证书**

[plugins."io.containerd.grpc.v1.cri".registry.configs]

[plugins."io.containerd.grpc.v1.cri".registry.configs."cof-lee.com:5443".tls]

insecure\_skip\_verify = true #跳过安全认证

#ca\_file="/etc/ssl/ca.cof-lee.com.crt" #设置ca证书

#cert\_file="/etc/ssl/cof-lee.com.crt" #设置服务器证书

[plugins."io.containerd.grpc.v1.cri".registry.configs."cof-lee.com:5443".auth]

username = "*userxx*"

password = "*passwdxx*"

**#添加一个默认镜像仓库**

[plugins."io.containerd.grpc.v1.cri".registry.mirrors]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."cof-lee.com:5443"]

endpoint = ["https://cof-lee.com:5443"]

/etc/containerd/config.toml 这个配置文件是给crictl和kubelet使用的，ctr不使用这个配置文件，因为ctr不使用CRI，所以它不读取 plugins."io.containerd.grpc.v1.cri" 配置。

**★解决方法，将服务器证书添加到操作系统的ca证书库里**

**①centos**

# cat ca.com.crt >> /etc/pki/tls/certs/ca-bundle.crt

#将ca证书添加到centos7系统证书信任列表中，链接到：

/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem

**②ubuntu**

# sudo cp myxxx.com.crt /usr/local/share/ca-certificates/ #复制要信任的证书到此目录下，只能以.crt结尾

# sudo update-ca-certificates #更新ca证书列表

# sudo cat /etc/ssl/certs/ca-certificates.crt #本质上是将要信任的证书内容写到此文件里

#或者在使用ctr命令下载镜像时指定--skip-verify参数

**★配置镜像保存位置**

# vi /etc/containerd/config.toml #添加或修改以下2行配置

root = "/containerd\_image" #镜像存放的位置，默认值为 "/var/lib/containerd"

state = "/run/containerd" #默认值，镜像运行时的数据存放目录

# mkdir /containerd\_image #不手动创建也行，重启containerd后会自动创建

# systemctl restart containerd #重启containerd

pull镜像时，镜像对应的config、index、layer、manifest都存储在以下默认目录中：

/var/lib/containerd/io.containerd.content.v1.content/blobs/sha256/

自定义root路径后，保存到以下目录中：

/containerd\_image/io.containerd.content.v1.content/blobs/sha256

控制台（标准输出）日志目录： /var/log/pods/容器名称/

**第2章、镜像操作**

# ctr images list #查看本地镜像

# ctr images ls #查看本地镜像

# ctr images ls -q #查看本地镜像，只显示镜像名称，不显示其他字段

# ctr images pull xxxxx.com/xxx/yyimg #下载镜像

# ctr images pull xxxxx.com/xxx/yyimg --skip-verify **#下载镜像时跳过ssl验证**

# ctr images pull xxxxx.com/xxx/yyimg --user=admin:passwdxx #下载镜像时指定用户名及密码

# ctr images tag xxxxx.com/xxx/yyimg cof-lee.com:5443/k8s/yyimg #打标签

# ctr images push cof-lee.com:5443/k8s/yyimg #上传镜像

# ctr images import nginx\_1.21.6.tar #导入镜像（默认default命名空间）

# ctr -n k8s.io images import nginx\_1.21.6.tar #导入镜像到指定的命名空间

# ctr images ls



REF这列表示镜像名称

# ctr images export xxx.tar imageName #导出镜像到xxx.tar文件

# ctr images rm xxxx #删除镜像，rm可换为delete, del, remove

**★注意**

①containerd不支持build,commit镜像

②containerd是有命名空间（namespace）的区分的，docker没有

# ctr namespaces list #查看命名空间（名称空间）

NAME LABELS

default #默认有个名为 default 的名称空间

镜像/容器的操作，若不指明名称空间，则默认在default的名称空间里操作

# ctr -n default images list #查看default名称空间里的镜像

# ctr --namespace default images list #查看default名称空间里的镜像

# ctr -n default images list #同上，命名空间参数只可跟在ctr后面

# ctr namespace create k8s.io #创建命名空间

**第3章、容器操作**

# ctr containers list #查看容器

# ctr containers ls #查看容器

# ctr -n default containers ls #查看default名称空间里的容器

#运行容器，-d表示后台运行，--net-host表示使用主机网络

# ctr run -d --net-host cof-lee.com:5443/k8s/nginx:1.21.6 nginxtest #最后的nginxtest为容器名称，可不指定

# ctr containers ls

CONTAINER IMAGE RUNTIME

nginxtest cof-lee.com:5443/k8s/nginx:1.21.6 io.containerd.runc.v2

# ctr containers info nginxtest #查看容器的详细信息

# ctr tasks ls #查看容器在宿主操作系统里的pid及状态

TASK PID STATUS

nginxtest 1965 RUNNING

# ctr tasks pause nginxtest #暂停容器

# ctr tasks resume nginxtest #继续运行容器

# ctr tasks kill --signal 9 nginxtest #停止容器

# ctr containers rm nginxtest #删除容器（可删除正在运行中的），rm可换为delete, del, remove

# ctr tasks rm -f nginxtest #删除容器，-f可删除正在运行中的容器，不带-f则只能删除已停止的

**第4章、容器运行时**

2013年Docker开源了容器镜像格式和运行时

2014年Google发布了Kubernetes的第一个Release版本，现已成长为容器编排领域的领导者

Docker的容器运行时解决方案采用两个核心技术：Namespace（资源隔离）和Cgroup（资源管理），

这两项技术其实在Docker之前早已进入Linux内核。

**★开放容器计划-OCI**

容器技术火了以后，Linux基金会带着一帮业界大佬向Docker~~施压~~发出合作邀请，最终Docker同意了，并于2015年6月在Docker大会DockerCon上推出容器标准，随后便成立了OCI（Open Container Initiative）开放容器计划组织，并发展成为Linux基金会下的一个项目。

OCI（Open Container Initialtive）提供了容器镜像和运行容器的规范。runc是OCI的一个实现，它是一个创建和运行容器进程的工具；runc实际上是从libcontainer演化过来的，并且是docker贡献给社区的第一个OCI参考实现，他就是用来创建和运行容器进行的工具；Docker也在其v1.11版本以后开始将runC作为自身服务的一个组件。

OCI标准目前包含两部分内容：

**容器运行时规范**： 该规范定义了如何根据相应的配置构建容器运行时。

**容器镜像规范**： 该规范定义了容器运行时使用的镜像的打包规范。

容器的运行时（runtime）就是运行和管理容器进程、镜像的工具，有containerd、rkt、Kata Container、CRI-O等。这些工具提供的功能不尽相同，有些只有容器运行的功能，有些除运行容器外还提供了容器镜像的管理功能。根据容器运行时提供功能，可以将容器运行时分为低层运行时和高层运行时。

**★低层运行时**

主要负责与宿主机操作系统打交道，根据指定的容器镜像在宿主机上运行容器的进程，并对容器的整个生命周期进行管理。设置容器的Namespace、Cgroups等基础操作

|  |  |  |
| --- | --- | --- |
| 低层运行时 |  |  |
| runc | 传统的运行时，基于Linux Namespace和Cgroups技术实现 |  |
| runv | 基于虚拟机管理程序的运行时，通过虚拟化 guest kernel，将容器和主机隔离开来，使得其边界更加清晰 | Kata Container  Firecracker |
| runsc | runc + safety ，通过拦截应用程序的所有系统调用，提供安全隔离的轻量级容器运行时沙箱 | gVisor |

**★高层运行时**

主要负责镜像的管理、转化等工作，为容器的运行做前提准备。主流的高层运行时主要有containerd和CRI-O。

容器运行之前一般先由高层运行时将容器镜像下载下来，并解压转换为容器运行需要的操作系统文件，再由低层运行时启动和管理容器。

**★容器运行时接口-CRI**

Kubernetes将对容器的操作抽象为一个接口，将接口作为kubelet与运行时工具之间的桥梁，kubelet通过发送接口请求对容器进行启动和管理，各个容器工具通过实现这个接口即可接入Kubernetes。这个统一的容器操作接口，就是CRI容器运行时接口（Container Runtime Interface）

CRI细分为两类接口：RuntimeService和ImageService

RuntimeService定义了和容器相关的操作，如创建、启动、删除容器等。

ImageService主要定义了容器镜像相关的操作，如拉取镜像、删除镜像等。

Kubernetes把docker shim内置在了官方的代码库中，将Docker设为Kubernetes默认的容器运行时工具。但是官方在Kubernetes 1.20版本的更新日志中声明已经废用对Docker的支持，并将在未来的版本中将其删除。在Kubernetes **1.24**版本中，dockershim代码已被删除，替换为containerd作为其默认运行时

CRI的出现为用户在底层容器运行时上提供了更多的选择，比如containerd和CRI-O。

contianerd来自于docker，它通过CRI插件来实现了CRI

CRI-O也是一个CRI的实现，它来自于Red Hat/IBM

oci是容器运行地的 规范

cri是调用容器运行时的 接口规范

podman官方未支持cri接口规范

**★docker与containerd的关系**

现在的docker与containerd不是同一个级别的东西，docker属于上层应用，containerd属于容器运行时

Docker最初是一个单体引擎，主要负责容器镜像的制作、上传、拉取及容器的运行及管理。随着容器技术的繁荣发展，为了促进容器技术相关的规范生成和Docker自身项目的发展，Docker将单体引擎拆分为三部分，分别为runC、containerd和dockerd

（在 2016 年 12 月 14 日，Docker 公司宣布将containerd 从 Docker 中分离，由开源社区独立发展和运营。Containerd 完全可以单独运行并管理容器，而 Containerd 的主要职责是镜像管理和容器执行。同时，Containerd 提供了 containerd-shim 接口封装层，向下继续对接 runC 项目，使得容器引擎 Docker Daemon 可以独立升级。）

其中runC主要负责容器的运行和生命周期的管理（低层运行时）

containerd主要负责容器镜像的下载和解压等镜像管理功能（高层运行时）

dockerd主要负责提供镜像制作、上传等功能同时提供容器存储和网络的映射功能，同时也是Docker服务器端的守护进程，用来响应Docker客户端（命令行CLI工具）发来的各种容器、镜像管理的任务。Docker公司将runC捐献给了OCI，将containerd捐献给了CNCF，剩下的dockerd作为Docker运行时由Docker公司自己维护。

k8s在1.23及之前版本默认是调用docker作为底层的容器运行时，从1.24版本开始移除了dockerShim组件，不再支持docker，从而默认使用containerd作为底层的容器运行时。

**k8s 1.23及之前版本：**

kubelet→dockerShim→dockerd→containerd→runC

默认调用的cri-socket： unix:///var/run/dockershim.sock

**k8s 1.24及之后版本：**

kubelet→cri-containerd→containerd→runC

后来cri-containerd重构进containerd中（CRI Plugin），合为一个containerd进程

默认调用的cri-socket： unix:///var/run/containerd/containerd.sock

**★第5章、ctr/nerdctl/crictl**

|  |  |  |  |
| --- | --- | --- | --- |
| 命令 | 安装包 | 功能差异 | 命名空间差异 |
| ctr | containerd.io | ctr为containerd自带的命名行工具；没有构建镜像的功能 | 支持-n指定命名空间；默认在default命名空间 |
| nerdctl | nerdctl | nerdctl可以操作containerd，和docker命令的语法一致；可构建镜像 | 支持-n指定命名空间；默认在default命名空间 |
| crictl | cri-tools | crictl为k8s的依赖工具，可以操作containred，没有tag命令，没有构建镜像的功能 | 默认在k8s.io命名空间，但没有-n参数，无法指定其他的命名空间 |

ctr -v输出的是containerd的版本，crictl -v输出的是当前cri-tools-xxxx.x86\_64.rpm的版本