|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
| **openEuler 1.0** | | |  |
| **容器用户指南** | | |
| **文档版本** | **01** | |
| **发布日期** | **2019-12-05** | |
|  | | | | |
|  | 华为技术有限公司 | |  |  |

|  |
| --- |
| 版权所有 © 华为技术有限公司2019。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | <https://www.huawei.com> |
| 客户服务邮箱： | [support@huawei.com](mailto:support@huawei.com) |
| 客户服务电话： | 4008302118 |

目 录

[1 概述 1](#_Toc26455430)

[2 安装与部署 2](#_Toc26455431)

[2.1 安装方法 2](#_Toc26455432)

[2.2 部署配置 2](#_Toc26455433)

[2.2.1 约束限制 2](#_Toc26455434)

[2.2.2 DAEMON多端口的绑定 3](#_Toc26455435)

[2.3 卸载 4](#_Toc26455436)

[3 使用方法 5](#_Toc26455437)

[3.1 容器管理 5](#_Toc26455438)

[3.1.1 创建容器 5](#_Toc26455439)

[3.1.2 启动容器 8](#_Toc26455440)

[3.1.3 运行容器 9](#_Toc26455441)

[3.1.4 停止容器 13](#_Toc26455442)

[3.1.5 删除容器 14](#_Toc26455443)

[3.1.6 接入容器 15](#_Toc26455444)

[3.1.7 重命名容器 16](#_Toc26455445)

[3.1.8 在容器中执行新命令 16](#_Toc26455446)

[3.1.9 查询单个容器信息 18](#_Toc26455447)

[3.1.10 查询所有容器信息 21](#_Toc26455448)

[3.1.11 重启容器 22](#_Toc26455449)

[3.1.12 等待容器退出 23](#_Toc26455450)

[3.1.13 查看容器中进程信息 24](#_Toc26455451)

[3.1.14 查看容器使用的资源 24](#_Toc26455452)

[3.1.15 获取容器日志 25](#_Toc26455453)

[3.1.16 容器与主机之间数据拷贝 26](#_Toc26455454)

[3.2 支持CNI网络 27](#_Toc26455455)

[3.2.1 描述 27](#_Toc26455456)

[3.2.2 接口 27](#_Toc26455457)

[3.2.2.1 CNI网络配置说明 28](#_Toc26455458)

[3.2.2.2 加入CNI网络列表 28](#_Toc26455459)

[3.2.2.3 退出CNI网络列表 29](#_Toc26455460)

[3.2.3 使用限制 29](#_Toc26455461)

[3.3 容器资源管理 29](#_Toc26455462)

[3.3.1 资源的共享 29](#_Toc26455463)

[3.3.2 限制运行时的CPU资源 31](#_Toc26455464)

[3.3.3 限制运行时的内存 32](#_Toc26455465)

[3.3.4 限制运行时的IO资源 33](#_Toc26455466)

[3.3.5 限制容器rootfs存储空间 34](#_Toc26455467)

[3.3.6 限制容器内文件句柄数 37](#_Toc26455468)

[3.3.7 限制容器内可以创建的进程/线程数 38](#_Toc26455469)

[3.3.8 配置容器内的ulimit值 39](#_Toc26455470)

[3.4 特权容器 41](#_Toc26455471)

[3.4.1 场景说明 41](#_Toc26455472)

[3.4.2 使用限制 41](#_Toc26455473)

[3.4.3 使用指导 43](#_Toc26455474)

[3.5 CRI接口 43](#_Toc26455475)

[3.5.1 描述 43](#_Toc26455476)

[3.5.2 接口 43](#_Toc26455477)

[3.5.2.1 Runtime服务 58](#_Toc26455478)

[3.5.2.1.1 RunPodSandbox 58](#_Toc26455479)

[3.5.2.1.2 StopPodSandbox 59](#_Toc26455480)

[3.5.2.1.3 RemovePodSandbox 59](#_Toc26455481)

[3.5.2.1.4 PodSandboxStatus 60](#_Toc26455482)

[3.5.2.1.5 ListPodSandbox 61](#_Toc26455483)

[3.5.2.1.6 CreateContainer 61](#_Toc26455484)

[3.5.2.1.7 StartContainer 62](#_Toc26455485)

[3.5.2.1.8 StopContainer 63](#_Toc26455486)

[3.5.2.1.9 RemoveContainer 63](#_Toc26455487)

[3.5.2.1.10 ListContainers 64](#_Toc26455488)

[3.5.2.1.11 ContainerStatus 64](#_Toc26455489)

[3.5.2.1.12 UpdateContainerResources 65](#_Toc26455490)

[3.5.2.1.13 ExecSync 66](#_Toc26455491)

[3.5.2.1.14 Exec 67](#_Toc26455492)

[3.5.2.1.15 Attach 67](#_Toc26455493)

[3.5.2.1.16 PortForward 68](#_Toc26455494)

[3.5.2.1.17 ContainerStats 68](#_Toc26455495)

[3.5.2.1.18 ListContainerStats 69](#_Toc26455496)

[3.5.2.1.19 UpdateRuntimeConfig 69](#_Toc26455497)

[3.5.2.1.20 Status 70](#_Toc26455498)

[3.5.2.2 Image服务 70](#_Toc26455499)

[3.5.2.2.1 ListImages 70](#_Toc26455500)

[3.5.2.2.2 ImageStatus 71](#_Toc26455501)

[3.5.2.2.3 PullImage 72](#_Toc26455502)

[3.5.2.2.4 RemoveImage 73](#_Toc26455503)

[3.5.2.2.5 ImageFsInfo 73](#_Toc26455504)

[3.5.3 约束 74](#_Toc26455505)

[3.6 镜像管理 75](#_Toc26455506)

[3.6.1 docker镜像管理 75](#_Toc26455507)

[3.6.1.1 从镜像仓库拉取镜像 75](#_Toc26455508)

[3.6.1.2 删除镜像 75](#_Toc26455509)

[3.6.1.3 加载镜像 76](#_Toc26455510)

[3.6.1.4 列出镜像 76](#_Toc26455511)

[3.6.1.5 检视镜像 76](#_Toc26455512)

[3.6.1.6 双向认证 77](#_Toc26455513)

[3.6.2 embedded镜像管理 78](#_Toc26455514)

[3.6.2.1 加载镜像 78](#_Toc26455515)

[3.6.2.2 列出镜像 79](#_Toc26455516)

[3.6.2.3 检视镜像 79](#_Toc26455517)

[3.6.2.4 删除镜像 80](#_Toc26455518)

[3.7 容器健康状态检查 80](#_Toc26455519)

[3.7.1 场景说明 80](#_Toc26455520)

[3.7.2 配置方法 80](#_Toc26455521)

[3.7.3 检查规则 81](#_Toc26455522)

[3.7.4 使用限制 81](#_Toc26455523)

[3.8 查询信息 82](#_Toc26455524)

[3.8.1 查询服务版本信息 82](#_Toc26455525)

[3.8.2 查询系统级信息 83](#_Toc26455526)

[3.9 安全特性 83](#_Toc26455527)

[3.9.1 seccomp安全配置场景 83](#_Toc26455528)

[3.9.1.1 场景说明 83](#_Toc26455529)

[3.9.1.2 使用限制 84](#_Toc26455530)

[3.9.1.3 使用指导 84](#_Toc26455531)

[3.9.2 capabilities安全配置场景 86](#_Toc26455532)

[3.9.2.1 场景说明 86](#_Toc26455533)

[3.9.2.2 使用限制 86](#_Toc26455534)

[3.9.2.3 使用指导 87](#_Toc26455535)

[3.10 支持OCI hooks 87](#_Toc26455536)

[3.10.1 描述 87](#_Toc26455537)

[3.10.2 接口 88](#_Toc26455538)

[3.10.3 使用限制 88](#_Toc26455539)

[4 附录 89](#_Toc26455540)

[4.1 命令行参数说明 89](#_Toc26455541)

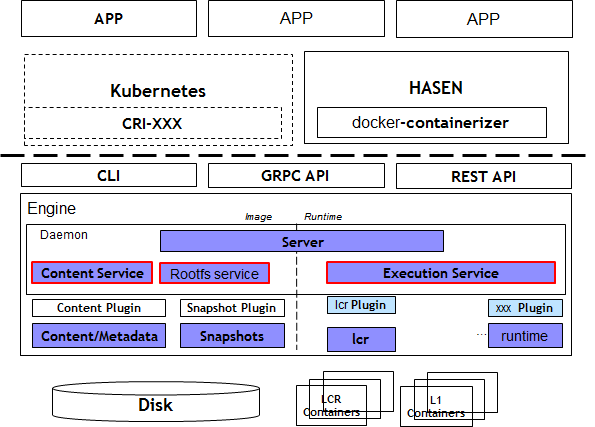
[4.2 CNI配置参数 91](#_Toc26455542)

# 概述

iSula通用容器引擎相比docker，是一种新的容器解决方案，提供统一的架构设计来满足CT和IT领域的不同需求。相比Golang编写的Docker，轻量级容器使用C/C++实现，具有轻、灵、巧、快的特点，不受硬件规格和架构的限制，底噪开销更小，可应用领域更为广泛。

容器统一架构如图1-1所示。

容器统一架构



# 安装与部署

[2.1 安装方法](#_ZH-CN_TOPIC_0184808040)

[2.2 部署配置](#_ZH-CN_TOPIC_0184808042)

[2.3 卸载](#_ZH-CN_TOPIC_0184808050)

## 安装方法

iSulad可以通过以下两种方式进行安装：

* 执行以下命令进行安装(推荐)：

$ sudo yum install -y iSulad

* 下载iSulad及其所有依赖库的RPM包手动安装：

$ sudo rpm -ihv *iSulad-xx.xx.xx-YYYYmmdd.HHMMSS.gitxxxxxxxx.openEulerxxx.aarch64*.rpm

## 部署配置

### 约束限制

* 高并发场景（并发启动200容器）下，glibc的内存管理机制会导致内存空洞以及虚拟内存较大（例如10GB）的问题。该问题是高并发场景下glibc内存管理机制的限制，而不是内存泄露，不会导致内存消耗无限增大。可以通过设置MALLOC\_ARENA\_MAX环境变量来减少虚拟内存的问题，而且可以增大减少物理内存的概率。但是这个环境变量会导致iSulad的并发性能下降，需要用户根据实际情况做配置。

参考实践情况，平衡性能和内存，可以设置MALLOC\_ARENA\_MAX为4。（在arm64服务器上面对iSulad的性能影响在10%以内）   
   
配置方法：   
1. 手动启动iSulad的场景，可以直接export MALLOC\_ARENA\_MAX=4，然后再启动iSulad即可。   
2. systemd管理iSulad的场景，可以修改/etc/sysconfig/iSulad，增加一条MALLOC\_ARENA\_MAX=4即可。

* 为daemon指定各种运行目录时的注意事项

以--root为例，当我们使用/new/path/作为daemon新的Root Dir时，如果/new/path/下已经存在文件，且目录或文件名与lcrd需要使用的目录或文件名冲突（例如：engines、mnt等目录）时，lcrd可能会更新原有目录或文件的属性，包括属主、权限等为自己的属主和权限。

所以，用户需要明白重新指定各种运行目录和文件，会对冲突目录、文件属性的影响。建议用户指定的新目录或文件为lcrd专用，避免冲突导致的文件属性变化以及带来的安全问题。

* 日志文件管理：



日志功能对接： iSulad由systemd管理，日志也由systemd管理，然后传输给rsyslogd。rsyslog默认会对写日志速度有限制，可以通过修改/etc/rsyslog.conf文件，增加"$imjournalRatelimitInterval 0"配置项，然后重启rsyslogd的服务即可。

* 命令行参数解析限制

使用iSulad命令行接口时，其参数解析方式与docker略有不同，对于命令行中带参数的flag，不管使用长flag还是短flag，只会将该flag后第一个空格或与flag直接相连接的'='后的字符串作为flag的参数，具体如下：

1) 使用短flag时，与破折号-连接的字符串中的每个字符都被当作短flag（当有=号时，=号后的字符串当成=号前的短flag的参数）。

lcrc run -du=root busybox 等价于 lcrc run -du root busybox 或 lcrc run -d -u=root busybox 或 lcrc run -d -u root busybox ，当使用lcrc run -du:root时，由于-:不是有效的短flag，因此会报错。前述的命令行也等价于lcrc run -ud root busybox，但不推荐这种使用方式，可能带来语义困扰。

2）使用长flag时，与双破折号--连接的字符串作为一个整体当成长flag，若包含=号，则=号前的字符串为长flag，=号后的为参数。

lcrc run --user=root busybox

等价于

lcrc run --user root busybox

* 启动一个lcrd容器，不能够以非root用户进行lcrc run -i/-t/-ti以及lcrc attach/exec操作。

### DAEMON多端口的绑定

描述

daemon端可以绑定多个unix socket或者tcp端口，并在这些端口上监听，客户端可以通过这些端口和daemon端进行交互。

接口

用户可以在/etc/isulad/daemon.json文件的hosts字段配置一个或者多个端口。当然用户也可以不指定hosts。

{   
 "hosts": [   
 "unix:///var/run/lcrd.sock",   
 "tcp://localhost:5678",   
 "tcp://127.0.0.1:6789"   
 ]   
}

用户也可以在/etc/sysconfig/iSulad中通过-H或者--host配置端口。用户同样可以不指定hosts。

OPTIONS='-H unix:///var/run/lcrd.sock --host tcp://127.0.0.1:6789'

如果用户在daemon.json文件及iSulad中均未指定hosts，则daemon在启动之后将默认监听unix:///var/run/lcrd.sock。

限制

* 用户不可以在/etc/isulad/daemon.json和/etc/sysconfig/iSuald两个文件中同时指定hosts，如果这样做将会出现错误，lcrd无法正常启动；

unable to configure the lcrd with file /etc/isulad/daemon.json: the following directives are specified both as a flag and in the configuration file: hosts: (from flag: [unix:///var/run/lcrd.sock tcp://127.0.0.1:6789], from file: [unix:///var/run/lcrd.sock tcp://localhost:5678 tcp://127.0.0.1:6789])

* 若指定的host是unix socket，则必须是合法的unix socket，需要以"unix://"开头，后跟合法的socket绝对路径；
* 若指定的host是tcp端口，则必须是合法的tcp端口，需要以"tcp://"开头，后跟合法的IP地址和端口，IP地址可以为localhost；
* 可以指定至多10个有效的端口，超过10个则会出现错误，lcrd无法正常启动。

## 卸载

1. 卸载安装包：

$ sudo yum remove iSulad

2. 镜像、容器、volumes以及相关配置文件不会自动删除，需要手动删除：

$ sudo rm -rf /var/lib/iSulad

# 使用方法

[3.1 容器管理](#_ZH-CN_TOPIC_0184808052)

[3.2 支持CNI网络](#_ZH-CN_TOPIC_0184808070)

[3.3 容器资源管理](#_ZH-CN_TOPIC_0184808077)

[3.4 特权容器](#_ZH-CN_TOPIC_0184808086)

[3.5 CRI接口](#_ZH-CN_TOPIC_0184808090)

[3.6 镜像管理](#_ZH-CN_TOPIC_0184808121)

[3.7 容器健康状态检查](#_ZH-CN_TOPIC_0184808136)

[3.8 查询信息](#_ZH-CN_TOPIC_0184808141)

[3.9 安全特性](#_ZH-CN_TOPIC_0184808145)

[3.10 支持OCI hooks](#_ZH-CN_TOPIC_0184808154)

## 容器管理

### 创建容器

描述

lcrc create 命令用于创建一个新的容器。容器引擎会使用指定的容器镜像创建容器读写层，或者使用指定的本地rootfs作为容器的运行环境。创建完成后，会将容器的ID输出到标准输出，后续可以使用lcrc start 命令启动该容器。新创建的容器状态为**inited**状态

用法

lcrc create [OPTIONS] IMAGE [COMMAND] [ARG...]

参数

create命令支持参数参考下表。

create命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **create** | --cap-drop | 删除Linux 权限功能 |
| --cgroup-parent | 指定容器cgroup父路径 |
| --cpuset-cpus | 允许执行的CPU（e.g. 0-3，0，1） |
| --cpu-shares | CPU份额（相对权重） |
| --cpu-quota | 限制CPU CFS（完全公平调度器）的配额 |
| --device=[] | 为容器添加一个主机设备 |
| -e, --env | 设置环境变量 |
| --entrypoint | 启动容器时要运行的入口点 |
| --external-rootfs=PATH | 指定一个不由LCRD管理的rootfs(可以为文件夹或块设备)给容器 |
| --files-limit | 调整容器内能够打开的文件句柄数（-1表示不限制） |
| --group-add=[] | 指定额外的用户组添加到容器 |
| --help | 打印帮助信息 |
| --hook-spec | 钩子配置文件 |
| -H, --host | 指定要连接的LCRD socket文件路径 |
| -h, --hostname | 容器主机名称 |
| -i, --interactive | 即使没有连接到容器的标准输入，也要保持容器的标准输入打开 |
| --hugetlb-limit=[] | 大页文件限制，例如：--hugetlb-limit 2MB:32MB |
| --log-opt=[] | 日志驱动程序选项，默认禁用记录容器串口日志功能，可以通过"--log-opt disable-log=false"来开启。 |
| -m, --memory | 内存限制 |
| --memory-reservation | 设置容器内存限制，默认与--memory一致。可认为--memory是硬限制，--memory-reservation是软限制；当使用内存超过预设值时，会动态调整（系统回收内存时尝试将使用内存降低到预设值以下），但不确保一定不超过预设值。一般可以和--memory一起使用，数值小于--memory的预设值，最小设置为4MB。 |
| --memory-swap | 正整数，内存 + 交换空间，-1 表示不限制 |
| --mount | 挂载主机目录到容器中 |
| --name=NAME | 容器名 |
| --net=none | 容器连接到网络 |
| --pids-limit | 调整容器内能够执行的进程数（-1表示不限制） |
| --privileged | 给予容器扩展的特权 |
| -R, --runtime | 容器运行时，参数支持"lcr"，忽略大小写，因此"LCR"和"lcr"是等价的 |
| --read-only | 设置容器的根文件系统为只读 |
| --restart | 当容器退出时重启策略  系统容器支持--restart on-reboot |
| --storage-opt | 配置容器的存储驱动选项 |
| -t, --tty | 分配伪终端 |
| --ulimit | 为容器设置ulimit限制 |
| -u, --user | 用户名或UID，格式[<name|uid>][:<group|gid>] |
| -v, --volume=[] | 挂载一个卷 |

约束限制

* 使用--user或--group-add参数，在容器启动阶段校验user或group时，容器如果使用的是OCI镜像，是从镜像的真实rootfs的etc/passwd和etc/group文件中校验，如果使用的是rootfs文件夹或块设备作为容器的rootfs，则校验的是host中的etc/passwd和etc/group文件；查找时使用的rootfs会忽略-v 和--mount等挂载参数，意味着使用这些参数尝试覆盖etc/passwd和etc/group两个文件时，在查找阶段不生效，只在容器真正启动时生效。生成的配置保存在"iSulad根目录/engine/容器ID/start\_generate\_config.json"，文件格式如下：

{   
 "uid": 0,   
 "gid": 8,   
 "additionalGids": [   
 1234,   
 8   
 ]   
}

示例

创建一个新容器

$ lcrc create busybox   
fd7376591a9c3d8ee9a14f5d2c2e5255b02cc44cddaabca82170efd4497510e1   
$ lcrc ps -a   
STATUS PID IMAGE COMMAND EXIT\_CODE RESTART\_COUNT STARTAT FINISHAT RUNTIME ID NAMES inited - busybox "sh" 0 0 - - lcr fd7376591a9c fd7376591a9c4521...

### 启动容器

描述

lcrc start命令用于启动一个或多个容器。

用法

lcrc start [OPTIONS] CONTAINER [CONTAINER...]

参数

start命令支持参数参考下表。

start命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **start** | -H, --host | 指定要连接的LCRD socket文件路径 |
| -R, --runtime | 容器运行时，参数支持"lcr"，忽略大小写，因此"LCR"和"lcr"是等价的 |

示例

启动一个新容器

$ lcrc start fd7376591a9c3d8ee9a14f5d2c2e5255b02cc44cddaabca82170efd4497510e1

### 运行容器

描述

lcrc run命令命令用于创建一个新的容器。会使用指定的容器镜像创建容器读写层，并且为运行指定的命令做好准备。创建完成后，使用指定的命令启动该容器。run命令相当于create然后start容器。

用法

lcrc run [OPTIONS] ROOTFS|IMAGE [COMMAND] [ARG...]

参数

run命令支持参数参考下表。

run命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **run** | --annotation | 设置容器的annotations。例如支持native.umask选项：  --annotation native.umask=normal # 启动的容器umask值为0022  --annotation native.umask=secure # 启动的容器umask值为0027  注意如果没有配置该参数，则使用lcrd中的umask配置。 |
| --cap-add | 添加Linux功能 |
| --cap-drop | 删除Linux功能 |
| --cgroup-parent | 指定容器cgroup父路径 |
| --cpuset-cpus | 允许执行的CPU（e.g. 0-3，0，1） |
| --cpu-shares | CPU份额（相对权重） |
| --cpu-quota | 限制CPU CFS（完全公平调度器）的配额 |
| -d, --detach | 后台运行容器并打印容器ID |
| --device=[] | 为容器添加一个主机设备 |
| -e, --env | 设置环境变量 |
| --entrypoint | 启动容器时要运行的入口点 |
| --external-rootfs=PATH | 指定一个不由LCRD管理的rootfs(可以为文件夹或块设备)给容器 |
| --files-limit | 调整容器内能够打开的文件句柄数（-1表示不限制） |
| --group-add=[] | 指定额外的用户组添加到容器 |
| --help | 打印帮助信息 |
| --hook-spec | 钩子配置文件 |
| -H, --host | 指定要连接的LCRD socket文件路径 |
| -h, --hostname | 容器主机名称 |
| --hugetlb-limit=[] | 大页文件限制，例如：--hugetlb-limit 2MB:32MB |
| -i, --interactive | 即使没有连接到容器的标准输入，也要保持容器的标准输入打开 |
| --log-opt=[] | 日志驱动程序选项，默认禁用记录容器串口日志功能，可以通过"--log-opt disable-log=false"来开启。 |
| -m, --memory | 内存限制 |
| --memory-reservation | 设置容器内存限制，默认与--memory一致。可认为--memory是硬限制，--memory-reservation是软限制；当使用内存超过预设值时，会动态调整（系统回收内存时尝试将使用内存降低到预设值以下），但不确保一定不超过预设值。一般可以和--memory一起使用，数值小于--memory的预设值，最小设置为4MB。 |
| --memory-swap | 正整数，内存 + 交换空间，-1 表示不限制 |
| --mount | 挂载主机目录到容器中 |
| --name=NAME | 容器名 |
| --net=none | 容器连接到网络 |
| --pids-limit | 调整容器内能够执行的进程数（-1表示不限制） |
| --privileged | 给予容器扩展的特权 |
| -R, --runtime | 容器运行时，参数支持"lcr"，忽略大小写，因此"LCR"和"lcr"是等价的 |
| --read-only | 设置容器的根文件系统为只读 |
| --restart | 当容器退出时重启策略  系统容器支持--restart on-reboot |
| --rm | 当容器退出时自动清理容器 |
| --storage-opt | 配置容器的存储驱动选项 |
| -t, --tty | 分配伪终端 |
| --ulimit | 为容器设置ulimit限制 |
| -u, --user | 用户名或UID，格式[<name|uid>][:<group|gid>] |
| -v, --volume=[] | 挂载一个卷 |

约束限制

* 容器父进程进程退出时，则对应的容器也自动退出。
* 创建普通容器时父进程不能为init，因为普通容器的权限不够，导致容器可以创建成功，但是attach进去的时候会卡住。
* 运行容器时，不指定--net，默认hostname为**localhost**。
* 使用--files-limit参数传入一个很小的值，如1时，启动容器时，iSulad创建cgroup子组后先设置files.limit值，再将容器进程的PID写入该子组的cgroup.procs文件，此时容器进程已经打开超过1个句柄，因而写入报错导致启动失败启动容器会失败。
* --mount参数和--volume参数同时存在时，如果目的路径有冲突，则--mount会在--volume之后挂载(即将--volume中的挂载点覆盖掉)。

备注：轻量级容器的参数中type支持bind或squashfs，当type=squashfs时，src是镜像的路径；原生docker的参数type支持bind、volume、tmpfs。

* restart重启策略不支持unless-stopped。
* 以下三种情况与docker 返回值不一致，docker返回127,轻量级容器返回125

--device参数指定主机设备为不存在的设备

--hook-spec参数指定不存在的hook json文件

--entrypoint 参数指定不存在的入口参数

* 禁止使用echo的方式向run命令的stdin输入数据，会导致客户端卡死。应该直接将echo的值作为命令行参数传给容器

[root@localhost ~]# echo ls | lcrc run -i busybox /bin/sh   
   
   
^C   
[root@localhost ~]#

上述命令出现客户端卡死现象，这是由于上述命令相当于往stdin输入ls，随后EOF被读取，客户端不再发送数据，等待服务端退出，但是服务端无法区分客户端是否需要继续发送数据，因而服务端卡在read数据上，最终导致双方均卡死。

正确的执行方式为：

[root@localhost ~]# lcrc run -i busybox ls   
bin   
dev   
etc   
home   
proc   
root   
sys   
tmp   
usr   
var   
[root@localhost ~]#

* 使用host的根目录（/）作为容器的文件系统，那么在挂载路径时，如果有如下情况

挂载情况

| Host 路径（source） | 容器路径（dest） |
| --- | --- |
| /home/test1 | /mnt/ |
| /home/test2 | /mnt/abc |



第一种情况，先挂载/home/test1，然后挂载/home/test2，这种情况会导致/home/test1的内容覆盖掉原来/mnt下面的内容，这样可能导致/mnt下面不存在abc目录，这样会导致挂载/home/test2到/mnt/abc失败。

第二种情况，先挂载/home/test2，然后挂载/home/test1。这种情况，第二次的挂载会把/mnt的内容替换为/home/test1的内容，这样第一次挂载的/home/test2到/mnt/abc的内容就看不到了。

因此，不支持第一种使用方式；第二种使用用户需要了解这种数据无法访问的风险



* 高并发场景（并发启动200容器）下，glibc的内存管理机制会导致内存空洞以及虚拟内存较大（例如10GB）的问题。该问题是高并发场景下glibc内存管理机制的限制，而不是内存泄露，不会导致内存消耗无限增大。可以通过设置MALLOC\_ARENA\_MAX环境变量来减少虚拟内存的问题，而且可以增大减少物理内存的概率。但是这个环境变量会导致iSulad的并发性能下降，需要用户根据实际情况做配置。

参考实践情况，平衡性能和内存，可以设置MALLOC\_ARENA\_MAX为4。（在arm64服务器上面对iSulad的性能影响在10%以内）   
   
配置方法：   
1. 手动启动iSulad的场景，可以直接export MALLOC\_ARENA\_MAX=4，然后再启动iSulad即可。   
2. systemd管理iSulad的场景，可以修改/etc/sysconfig/iSulad，增加一条MALLOC\_ARENA\_MAX=4即可。

示例

运行一个新容器

$ lcrc run -itd busybox   
9c2c13b6c35f132f49fb7ffad24f9e673a07b7fe9918f97c0591f0d7014c713b

### 停止容器

描述

lcrc stop命令用于停止一个或多个运行中的容器。首先向容器中的首进程会发送**SIGTERM**信号，在指定时间（默认为10s）内容器未停止时，会发送**SIGKILL**。

用法

lcrc stop [OPTIONS] CONTAINER [CONTAINER...]

参数

stop命令支持参数参考下表。

stop命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **stop** | -f, --force | 强制停止正在运行的容器 |
| -H, --host | 指定要连接的LCRD socket文件路径 |
| -t, --time | 先优雅停止，超过这个时间，则强行终止 |

约束限制

* 指定t参数且t<0时，请确保自己容器的应用会处理stop信号。

Stop的原理：

Stop会首先给容器发送Stop 信号（SIGTERM）

然后等待一定的时间（这个时间就是用户输入的 t）

过了指定时间，如果容器还仍处于运行状态，那么就发送kill信号（SIGKILL），强杀。

输入参数t的含义：

t<0 : 表示一直等待，不管多久都等待程序优雅退出，既然用户这么输入了，表示对自己的应用比较放心，认为自己的程序有 合理的stop信号的处理机制。

t=0 ： 表示不等，立即发送kill -9 到容器。

t>0 ： 表示等一定的时间，如果容器还未退出，就发送kill -9 到容器。

所以如果用户使用t<0 (比如t=-1)，请确保自己容器的应用会正确处理SIGTERM. 如果容器忽略了该信号，会导致lcrc stop一直卡住。

示例

停止一个容器

$ lcrc stop fd7376591a9c3d8ee9a14f5d2c2e5255b02cc44cddaabca82170efd4497510e1   
fd7376591a9c3d8ee9a14f5d2c2e5255b02cc44cddaabca82170efd4497510e1

### 删除容器

描述

lcrc rm命令用于删除一个或多个容器。

用法

lcrc rm [OPTIONS] CONTAINER [CONTAINER...]

参数

rm命令支持参数参考下表。

rm命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **rm** | -f, --force | 强制移除正在运行的容器 |
| -H, --host | 指定要连接的LCRD socket文件路径 |
| -v, --volume | 移除挂载在容器上的卷（备注：目前LCRD尚不使用此功能） |

约束限制

* 在IO正常情况，空环境（只有1个容器）删除一个running容器的时间为T1，200个容器的环境（容器无大量IO操作，host IO正常）删除一个running容器所需时间为T2。T2的规格为：T2 = max { T1 \* 3, 5 } 秒钟。

示例

删除一个停止状态的容器

$ lcrc rm fd7376591a9c3d8ee9a14f5d2c2e5255b02cc44cddaabca82170efd4497510e1   
fd7376591a9c3d8ee9a14f5d2c2e5255b02cc44cddaabca82170efd4497510e1

### 接入容器

描述

lcrc attach命令用于将当前终端的标准输入、标准输出和标准错误连接到正在运行的容器。

用法

lcrc attach [OPTIONS] CONTAINER

参数

attach命令支持参数参考下表。

attach命令参数列表

| **命令** | **参数** | **说明** |
| --- | --- | --- |
| **attach** | --help | 打印帮助信息 |
| -H, --host | 指定要连接的LCRD socket文件路径 |
| -D, --debug | 开启debug模式 |

约束限制

* 原生docker attach容器会直接进入容器，而isulad attach容器后需要敲一个回车才进入。

示例

接入一个运行状态的容器

$ lcrc attach fd7376591a9c3d8ee9a14f5d2c2e5255b02cc44cddaabca82170efd4497510e1   
/ #   
/ #

### 重命名容器

描述

lcrc rename命令用于重命名容器。

用法

lcrc rename [OPTIONS] OLD\_NAME NEW\_NAME

参数

rename命令支持参数参考下表。

rename 命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **rename** | -H, --host | 重命名容器 |

示例

重命名一个容器

$ lcrc rename my\_container my\_new\_container

### 在容器中执行新命令

描述

lcrc exec命令用于正在运行的容器中运行一个新命令。新执行的命令将在容器的默认目录中运行。如果基础镜像指定了自定义目录，则将使用该目录。

用法

lcrc exec [OPTIONS] CONTAINER COMMAND [ARG...]

参数

exec命令支持参数参考下表。

exec命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **exec** | -d, --detach | 后台运行命令 |
| -e, --env | 设置环境变量（备注：目前LCRD尚不使用此功能） |
| -H, --host | 指定要连接的LCRD socket文件路径 |
| -i, --interactive | 没有连接，也要保持标准输入打开（备注：目前LCRD尚不使用此功能） |
| -t, --tty | 分配伪终端（备注：目前LCRD尚不使用此功能） |

约束限制

* lcrc exec 不指定任何参数时，会默认使用-it参数， 表示分配一个伪终端，以交互式的方式进入容器
* 当使用lcrc exec 执行脚本，在脚本中执行后台进程时，需使用nohup标志忽略SIGHUP信号。

使用lcrc exec运行脚本,在脚本中运行后台进程需使用nohup标志。否则内核会在exec执行的进程（session首进程）退出时，向后台执行的进程发送SIGHUP信号，导致后台进程退出，出现僵尸进程。

* lcrc exec 进入容器进程后，不能执行后台程序，否则会出现卡死现象。

lcrc exec执行后台进程的方式如下：

* 1. 使用lcrc exec进入容器终端，lcrc exec container\_name bash
  2. 进入容器后，执行 script &
  3. 执行exit，导致终端卡死

lcrc exec 进入容器后，执行后台程序卡住的原因为lcrc exec进入容器运行后台while1程序，当bash退出时，while1程序并不会退出，变为孤儿进程由1号   
进程接管，while1程序是由容器的初始bash进程fork &exec执行的，while1进程复制了bash进程的文件句柄，导致bash退出时，句柄并未完全关闭，导致   
console进程收不到句柄关闭事件，epoll\_wait卡住，进程不退出。

* lcrc exec 不能用后台方式执行，否则可能会出现卡死现象。

lcrc exec后台执行的方式如下：

使用**lcrc exec 脚本 &** 的方式后台执行exec，如：lcrc exec container\_name script & ，lcrc exec 后台执行，执行的脚本中不断cat某一文件，正常时在当前终端有输出，如果在当前终端执行回车操作，客户端会因读IO失败而退出读stdout的动作，使终端不再输出，服务端由于进程仍然在cat文件，会继续往fifo的buffer里写入数据，当缓存写满时，容器内进程会卡死在write动作上。

* 轻量级容器使用exec执行带有管道操作的命令时，建议使用/bin/bash -c 方式执行该命令。

典型应用场景：

使用lcrc exec container\_name -it ls /test | grep "xx" | wc -l,用于统计test目录下xx的文件个数，因exec执行的为"ls /test",其输出通过管道进行grep、wc 处理。exec执行的为"ls /test"的输出会换行，再针对该输出进行处理时，结果有误。

原因：使用exec 执行ls /test,输出带有换行，针对该输出进行“| grep "xx" | wc -l“，处理结果为2（两行）

[root@localhost ~]# lcrc exec -it container ls /test   
xx xx10 xx12 xx14 xx3 xx5 xx7 xx9   
xx1 xx11 xx13 xx2 xx4 xx6 xx8   
[root@localhost ~]#

建议处理方式：使用run/exec执行带有管道操作的命令时，使用/bin/bash -c 执行命令，在容器中执行管道操作。

[root@localhost ~]# lcrc exec -it container /bin/sh -c "ls /test | grep "xx" | wc -l"   
15   
[root@localhost ~]#

* 禁止使用echo的方式向exec命令的stdin输入数据，会导致客户端卡死。应该直接将echo的值作为命令行参数传给容器

[root@localhost ~]# echo ls | lcrc exec 38 /bin/sh   
   
   
^C   
[root@localhost ~]#

上述命令可能出现客户端卡死现象，这是由于上述命令相当于往stdin输入ls，随后EOF被读取，客户端不再发送数据，等待服务端退出，但是服务端无法区分客户端是否需要继续发送数据，因而服务端卡在read数据上，最终导致双方均卡死。

正确的执行方式为：

[root@localhost ~]# lcrc exec 38 ls   
bin dev etc home proc root sys tmp usr var

示例

在运行中的容器中，执行echo命令

$ lcrc exec c75284634bee echo "hello,world"   
hello,world

### 查询单个容器信息

描述

lcrc inspect提供了容器的详细信息。

用法

lcrc inspect [OPTIONS] CONTAINER|IMAGE [CONTAINER|IMAGE...]

参数

inspect命令支持参数参考下表。

inspect命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **inspect** | -H, --host | 指定要连接的LCRD socket文件路径 |
| -f, --format | 使用模板格式化输出 |
| -t, --time | 超时时间的秒数，若在该时间内inspect查询容器信息未执行成功，则停止等待并立即报错，默认为120秒，当配置小于等于0的值，表示不启用timeout机制inspect查询容器信息会一直等待，直到获取容器信息成功后返回。 |

约束限制

* 轻量级容器不支持format为”{{.State}}”的格式化输出，提示只支持”{{json .State}}”的json格式化输出。当inspect镜像时，不支持-f参数。

示例

查询容器信息

$ lcrc inspect c75284634bee   
[   
 {   
 "Id": "c75284634beeede3ab86c828790b439d16b6ed8a537550456b1f94eb852c1c0a",   
 "Created": "2019-08-01T22:48:13.993304927-04:00",   
 "Path": "sh",   
 "Args": [],   
 "State": {   
 "Status": "running",   
 "Running": true,   
 "Paused": false,   
 "Restarting": false,   
 "Pid": 21164,   
 "ExitCode": 0,   
 "Error": "",   
 "StartedAt": "2019-08-02T06:09:25.535049168-04:00",   
 "FinishedAt": "2019-08-02T04:28:09.479766839-04:00",   
 "Health": {   
 "Status": "",   
 "FailingStreak": 0,   
 "Log": []   
 }   
 },   
 "Image": "busybox",   
 "ResolvConfPath": "",   
 "HostnamePath": "",   
 "HostsPath": "",   
 "LogPath": "none",   
 "Name": "c75284634beeede3ab86c828790b439d16b6ed8a537550456b1f94eb852c1c0a",   
 "RestartCount": 0,   
 "HostConfig": {   
 "Binds": [],   
 "NetworkMode": "",   
 "GroupAdd": [],   
 "IpcMode": "",   
 "PidMode": "",   
 "Privileged": false,   
 "SystemContainer": false,   
 "NsChangeFiles": [],   
 "UserRemap": "",   
 "ShmSize": 67108864,   
 "AutoRemove": false,   
 "AutoRemoveBak": false,   
 "ReadonlyRootfs": false,   
 "UTSMode": "",   
 "UsernsMode": "",   
 "Sysctls": {},   
 "Runtime": "lcr",   
 "RestartPolicy": {   
 "Name": "no",   
 "MaximumRetryCount": 0   
 },   
 "CapAdd": [],   
 "CapDrop": [],   
 "Dns": [],   
 "DnsOptions": [],   
 "DnsSearch": [],   
 "ExtraHosts": [],   
 "HookSpec": "",   
 "CPUShares": 0,   
 "Memory": 0,   
 "OomScoreAdj": 0,   
 "BlkioWeight": 0,   
 "BlkioWeightDevice": [],   
 "CPUPeriod": 0,   
 "CPUQuota": 0,   
 "CPURealtimePeriod": 0,   
 "CPURealtimeRuntime": 0,   
 "CpusetCpus": "",   
 "CpusetMems": "",   
 "SecurityOpt": [],   
 "StorageOpt": {},   
 "KernelMemory": 0,   
 "MemoryReservation": 0,   
 "MemorySwap": 0,   
 "OomKillDisable": false,   
 "PidsLimit": 0,   
 "FilesLimit": 0,   
 "Ulimits": [],   
 "Hugetlbs": [],   
 "HostChannel": {   
 "PathOnHost": "",   
 "PathInContainer": "",   
 "Permissions": "",   
 "Size": 0   
 },   
 "EnvTargetFile": "",   
 "ExternalRootfs": ""   
 },   
 "Mounts": [],   
 "Config": {   
 "Hostname": "localhost",   
 "User": "",   
 "Env": [   
 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",   
 "TERM=xterm",   
 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"   
 ],   
 "Tty": true,   
 "Cmd": [   
 "sh"   
 ],   
 "Entrypoint": [],   
 "Labels": {},   
 "Annotations": {   
 "log.console.file": "none",   
 "log.console.filerotate": "7",   
 "log.console.filesize": "1MB",   
 "rootfs.mount": "/var/lib/lcrd/mnt/rootfs",   
 "native.umask": "secure"   
 },   
 "HealthCheck": {   
 "Test": [],   
 "Interval": 0,   
 "Timeout": 0,   
 "StartPeriod": 0,   
 "Retries": 0,   
 "ExitOnUnhealthy": false   
 }   
 },   
 "NetworkSettings": {   
 "IPAddress": ""   
 }   
 }   
]

### 查询所有容器信息

描述

lcrc ps 用于查询所有容器的信息。

用法

lcrc ps [OPTIONS]

参数

ps命令支持参数参考下表。

ps命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **ps** | -a, --all | 显示所有的容器 |
| -H, --host | 指定要连接的LCRD socket文件路径 |
| -q, --quiet | 只显示容器名字 |
| -f, --filter | 增加筛选过滤条件 |

示例

查询所有容器信息

$ lcrc ps -a   
STATUS PID IMAGE COMMAND EXIT\_CODE RESTART\_COUNT STARTAT FINISHAT RUNTIME ID NAMES running 21164 busybox "sh" 0 0 23 minutes ago 2 hours ago lcr c75284634bee c75284634beeede3ab86c828790b439d16b6ed8a537550456b1f94eb852c1c0a

### 重启容器

描述

lcrc restart 用于重启一个或者多个容器。

用法

lcrc restart [OPTIONS] CONTAINER [CONTAINER...]

参数

restart命令支持参数参考下表。

restart 命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **restart** | -H, --host | 指定要连接的LCRD socket文件路径 |
| -t, --time | 先优雅停止，超过这个时间，则强行终止 |

约束限制

* 指定t参数且t<0时，请确保自己容器的应用会处理stop信号。

restart 会首先调用stop停止容器：

stop会首先给容器发送stop信号（SIGTERM）

然后等待一定的时间（这个时间就是用户输入的 t）

过了一定时间，如果容器仍处于运行状态，那么就发送kill信号（SIGKILL），强杀。

输入参数t的含义：

t<0 : 表示一直等待，不管多久都等待程序优雅退出，既然用户这么输入了，表示对自己的应用比较放心，认为自己的程序有合理的stop信号的处理机制。

t=0 ： 表示不等，立即发送kill -9 到容器。

t>0 ： 表示等一定的时间，如果容器还未退出，就发送kill -9 到容器。

所以如果用户使用t<0 (比如t=-1)，请确保自己容器的应用会正确处理SIGTERM. 如果容器忽略了该信号，会导致lcrc restart一直卡住。

示例

重启单个容器

$ lcrc restart c75284634beeede3ab86c828790b439d16b6ed8a537550456b1f94eb852c1c0a   
 c75284634beeede3ab86c828790b439d16b6ed8a537550456b1f94eb852c1c0a

### 等待容器退出

描述

lcrc wait用于等待一个或者多个容器退出。

用法

lcrc wait [OPTIONS] CONTAINER [CONTAINER...]

参数

wait命令支持参数参考下表。

wait命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **wait** | -H, --host | 指定要连接的LCRD socket文件路径 |
| / | 阻塞，直到容器停止，然后打印退出代码 |

示例

重启单个容器

$ lcrc wait c75284634beeede3ab86c828790b439d16b6ed8a537550456b1f94eb852c1c0a   
 137

### 查看容器中进程信息

描述

lcrc top用于查看容器中的进程信息。

用法

lcrc top [OPTIONS] container [ps options]

参数

top命令支持参数参考下表。

top命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **top** | -H, --host | 指定要连接的LCRD socket文件路径 |
| / | 查询运行容器的进程信息 |

示例

查询容器中进程信息

$ lcrc top 21fac8bb9ea8e0be4313c8acea765c8b4798b7d06e043bbab99fc20efa72629c   
UID PID PPID C STIME TTY TIME CMD   
root 22166 22163 0 23:04 pts/1 00:00:00 sh

### 查看容器使用的资源

描述

lcrc stats用于实时显示资源使用的统计信息。

用法

lcrc stats [OPTIONS] [CONTAINER...]

参数

stats命令支持参数参考下表。

stats命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **stats** | -H, --host | 指定要连接的LCRD socket文件路径 |
| -a, --all | 显示所有容器（默认只显示运行中的容器） |
| --no-stream | 非流式方式的stats，只打印第一次结果 |

示例

显示资源使用的统计信息

$ lcrc stats --no-stream 21fac8bb9ea8e0be4313c8acea765c8b4798b7d06e043bbab99fc20efa72629c CONTAINER CPU % MEM USAGE / LIMIT MEM % BLOCK I / O PIDS   
21fac8bb9ea8 0.00 56.00 KiB / 7.45 GiB 0.00 0.00 B / 0.00 B 1

### 获取容器日志

描述

lcrc logs用于获取容器的日志。

用法

lcrc logs [OPTIONS] [CONTAINER...]

参数

logs命令支持参数参考下表。

logs命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **logs** | -H, --host | 指定要连接的LCRD socket文件路径 |
| -f, --follow | 跟踪日志输出 |
| --tail | 显示日志行数 |

约束限制

* 容器串口logs日志记录功能，默认为关闭状态，需要开启可以通过 lcrc create --log-opt disable-log=false 或 lcrc run --log-opt disable-log=false 开启。

示例

获取容器日志

$ lcrc logs 6a144695f5dae81e22700a8a78fac28b19f8bf40e8827568b3329c7d4f742406   
hello, world   
hello, world   
hello, world

### 容器与主机之间数据拷贝

描述

lcrc cp 用于容器与主机之间的数据拷贝。

用法

lcrc cp [OPTIONS] CONTAINER:SRC\_PATH DEST\_PATH   
lcrc cp [OPTIONS] SRC\_PATH CONTAINER:DEST\_PATH

参数

cp命令支持参数参考下表。

cp命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **cp** | -H, --host | 指定要连接的LCRD socket文件路径 |

约束限制

* iSulad在执行拷贝时，不会挂载/etc/hostname， /etc/resolv.conf，/etc/hosts三个文件，也不会对--volume和--mount参数传入的参数挂载到host，所以对这些文件的拷贝使用的是镜像中的原始文件，而不是真实容器中的文件。

[root@localhost tmp]# lcrc cp b330e9be717a:/etc/hostname /tmp/hostname   
[root@localhost tmp]# cat /tmp/hostname   
[root@localhost tmp]#

* iSulad在解压文件时，不会对文件系统中即将被覆盖的文件或文件夹做类型判断，而是直接覆盖，所以在拷贝时，如果源为文件夹，同名的文件会被强制覆盖为文件夹；如果源为文件，同名的文件夹会被强制覆盖为文件。

[root@localhost tmp]# rm -rf /tmp/test\_file\_to\_dir && mkdir /tmp/test\_file\_to\_dir   
[root@localhost tmp]# lcrc exec b330e9be717a /bin/sh -c "rm -rf /tmp/test\_file\_to\_dir && touch /tmp/test\_file\_to\_dir"   
[root@localhost tmp]# lcrc cp b330e9be717a:/tmp/test\_file\_to\_dir /tmp   
[root@localhost tmp]# ls -al /tmp | grep test\_file\_to\_dir   
-rw-r----- 1 root root 0 Apr 26 09:59 test\_file\_to\_dir

示例

将主机/test/host目录拷贝到容器21fac8bb9ea8的/test目录下。

lcrc cp /test/host 21fac8bb9ea8:/test

将容器21fac8bb9ea8的/www目录拷贝到主机的/tmp目录中。

lcrc cp 21fac8bb9ea8:/www /tmp/

## 支持CNI网络

### 描述

实现CRI接口对接CNI网络的能力，包括CNI网络配置文件的解析、CNI网络的加入和退出。Pod需要支持网络时，例如通过canal等容器网络插件提供网络能力，那么需要CRI接口能够和canal实现对接，并且调用canal的接口，为Pod提供网络能力。

### 接口

CNI对用户可见的接口，主要涉及CNI网络配置和Pod配置中CNI网络相关的项。

* CNI网络配置相关的接口，主要是isulad指定CNI网络配置文件所在路径、CNI网络插件二进制文件所在的路径以及使用的网络模式；表3-18。
* Pod配置中CNI网络相关的项，主要是设置Pod加入的附加CNI网络列表，默认情况Pod只会加入到default CNI网络平面中，可以通过该配置把Pod加入到多个CNI网络平面中。

CNI网络配置接口

|  | 命令行 | 配置文件 | 说明 |
| --- | --- | --- | --- |
| 设置CNI网络插件二进制文件所在路径 | --cni-bin-dir | "cni-bin-dir": "", | 默认为/opt/cni/bin |
| 设置CNI网络配置文件所在路径 | --cni-conf-dir | "cni-conf-dir": "", | 系统会遍历目录下面所有后缀名为".conf"、".conflist"和 ".json"的文件。默认为/etc/cni/net.d |
| 指定网络模式 | --network-plugin | "network-plugin": "", | 指定网络插件，默认为空字符，表示无网络配置，创建的sandbox只有loop网卡。支持cni和空字符，其他非法值会导致lcrd启动失败。 |

附加CNI网络配置方式：

在Pod的配置文件的"annotations"中，增加一项"network.alpha.kubernetes.io/network": "网络平面配置"；

网络平面配置为json格式，包含两项：

* name：指定CNI网络平面的名字
* interface：指定网络接口的名字

附件CNI网络配置方式示例如下：

"annotations" : {   
 "network.alpha.kubernetes.io/network": "{\"name\": \"mynet\", \"interface\": \"eth1\"}"   
 }

#### CNI网络配置说明

CNI网络配置包含两种类型，文件格式都为json：

* 单网络平面配置，以.conf和.json为后缀的文件：具体的配置项请参见附录的表4-8。
* 多网络平面配置，以.conflist为后缀的文件：具体的配置项请参见附录的表4-10。

#### 加入CNI网络列表

如果iSulad配置了--network-plugin=cni，而且设置了default网络平面配置，那么在启动Pod的时候，会自动把Pod加入到default网络平面。如果在Pod的配置中配置了附加网络配置，那么启动Pod的时候也会把Pod加入到这些附加网络平面中。

Pod配置中和网络相关的还有port\_mappings项，用于设置Pod的端口映射关系。配置方式如下：

"port\_mappings":[   
 {   
 "protocol": 1,   
 "container\_port": 80,   
 "host\_port": 8080   
 }   
]

* protocal：表示映射使用的协议，支持tcp（用0标识）、udp（用1标识）；
* container\_port：表示容器映射出去的port；
* host\_port：表示映射到主机的port。

#### 退出CNI网络列表

StopPodSandbox的时候，会调用退出CNI网络的接口，清理网络相关的资源。



1. 在调用RemovePodSandbox接口之前，至少要调用一次StopPodSandbox接口

2. StopPodSandbox调用CNI接口失败，导致的网络资源残留，由canal组件负责清理。

### 使用限制

* cniVersion的版本，当前只支持0.3.0和0.3.1。由于后期可能需要支持0.1.0和0.2.0，错误日志打印时，保留了0.1.0和0.2.0的提示信息。
* name：必须是小写字符、数字、'-'以及'.'组成； '.'和'-'不能作为首字符和尾字符； 而且长度不超过200个字符。
* 配置文件个数不超过200个，单个配置文件大小不超过1MB。
* 扩展之后的参数，需要根据实际网络需求来配置，不需要使用的可选参数可以不写入到netconf.json文件中。

## 容器资源管理

### 资源的共享

描述

容器间或者容器与host之间可以共享namespace信息，包括pid, net, ipc, uts。

用法

lcrc create/run时使用namespace相关的参数共享资源，具体参数见下方参数列表。

参数

create/run时可以指定下列参数。

| **参数项** | **参数说明** | **取值范围** | **是否必选** |
| --- | --- | --- | --- |
| --pid | 指定要共享的pid namespace | [none, host, container:<containerID>]，none表示不共享，host表示与host共用namespace，container:<containerID>表示与容器containerID共享同一个namespace | 否 |
| --net | 指定要共享的net namespace | [none, host, container:<containerID>]，none表示不共享，host表示与host共用namespace，container:<containerID>表示与容器containerID共享同一个namespace | 否 |
| --ipc | 指定要共享的ipc namespace | [none, host, container:<containerID>]，none表示不共享，host表示与host共用namespace，container:<containerID>表示与容器containerID共享同一个namespace | 否 |
| --uts | 指定要共享的uts namespace | [none, host, container:<containerID>]，none表示不共享，host表示与host共用namespace，container:<containerID>表示与容器containerID共享同一个namespace | 否 |

示例

如果两个容器需要共享同一个pid namespace，在运行容器时，直接加上--pid container:<containerID> 即可，如：

lcrc run -tid --name test\_pid busybox sh   
lcrc run -tid --name test --pid container:test\_pid busybox sh

### 限制运行时的CPU资源

描述

可以通过参数限制容器的各项cpu资源值。

用法

lcrc create/run时使用cpu相关的参数限制容器的各项cpu资源值，具体参数及取值见下方参数列表。

参数

create/run时可以指定下列参数。

| **参数项** | **参数说明** | **取值范围** | **是否必选** |
| --- | --- | --- | --- |
| --cpu-period | 限制容器中cpu cfs（完全公平调度）周期 | 64位整数(int64) | 否 |
| --cpu-quota | 限制容器中cpu cfs(完全公平调度) 的配额 | 64位整数(int64) | 否 |
| --cpu-shares | 限制容器中cpu相对权重 | 64位整数(int64) | 否 |
| --cpuset-cpus | 限制容器中使用cpu节点 | 字符串。值为要设置的cpu编号，有效范围为主机上的cpu数量，例如可以设置0-3或者0,1. | 否 |
| --cpuset-mems | 限制容器中cpuset使用的mem节点 | 字符串。值为要设置的cpu编号，有效范围为主机上的cpu数量，例如可以设置0-3或者0,1. | 否 |

示例

如果需要限制容器只是用特定的cpu，在运行容器时，直接加上--cpuset-cpus number 即可，如：

lcrc run -tid --cpuset-cpus 0,2-3 busybox sh



是否设置成功，请参见“查询单个容器信息”章节。

### 限制运行时的内存

描述

可以通过参数限制容器的各项内存值上限。

用法

lcrc create/run时使用内存相关的参数限制容器的各项内存使用上限，具体参数及取值见下方参数列表。

参数

create/run时可以指定下列参数。

| **参数项** | **参数说明** | **取值范围** | **是否必选** |
| --- | --- | --- | --- |
| --memory | 限制容器中内存使用上限 | 64位整数(int64)。值为非负数，0表示不设置（不限制）；单位可以为空(byte)，KB，MB，GB，TB，PB. | 否 |
| --memory-reservation | 限制容器中内存的软上限 | 64位整数(int64)。值为非负数，0表示不设置（不限制）；单位可以为空(byte)，KB，MB，GB，TB，PB. | 否 |
| --memory-swap | 限制容器中交换内存的上限 | 64位整数(int64)。值为-1或非负数，-1表示不限制，0表示不设置（不限制）；单位可以为空(byte)，KB，MB，GB，TB，PB. | 否 |
| --kernel-memory | 限制容器中内核内存的上限 | 64位整数(int64)。值为非负数，0表示不设置（不限制）；单位可以为空(byte)，KB，MB，GB，TB，PB. | 否 |

示例

如果需要限制容器内内存的上限，在运行容器时，直接加上--memory <number>[<unit>]即可，如：

lcrc run -tid --memory 1G busybox sh

### 限制运行时的IO资源

描述

可以通过参数限制容器中设备读写速度。

用法

lcrc create/run时使用--device-read-bps/--device-write-bps <device-path>:<number>[<unit>]来限制容器中设备的读写速度。

参数

create/run时指定--device-read/write-bps参数。

| **参数项** | **参数说明** | **取值范围** | **是否必选** |
| --- | --- | --- | --- |
| --device-read-bps/--device-write-bps | 限制容器中设备的读速度/写速度 | 64位整数(int64)。值为正整数，可以为0，0表示不设置（不限制）；单位可以为空(byte)，KB，MB，GB，TB，PB. | 否 |

示例

如果需要限制容器内设备的读写速度，在运行容器时，直接加上--device-write-bps/--device-read-bps <device-path>:<number>[<unit>]即可，例如，限制容器busybox内设备/dev/sda的读速度为 1MB 每秒，则命令如下：

lcrc run -tid --device-write /dev/sda:1mb busybox sh

限制写速度的命令如下：

lcrc run -tid read-bps /dev/sda:1mb busybox sh

### 限制容器rootfs存储空间

描述

在ext4上使用overlay2时，可以设置单个容器的文件系统限额，比如设置A容器的限额为5G，B容器为10G。

该特性通过ext4文件系统的project quota功能来实现，在内核支持的前提下，通过系统调用SYS\_IOCTL设置某个目录的project ID，再通过系统调用SYS\_QUOTACTL设置相应的project ID的hard limit和solft limit值达到限额的目的。

用法

1. 环境准备

文件系统支持Project ID和Project Quota属性，4.19版本内核已经支持，外围包e2fsprogs版本不低于1.43.4-2。

1. 在容器挂载overlayfs之前，需要对不同容器的upper目录和work目录设置不同的project id，同时设置继承选项，在容器挂载overlayfs之后不允许再修改project id和继承属性。
2. 配额的设置需要在容器外以特权用户进行。
3. daemon中增加如下配置

-s overlay2 --storage-opt overlay2.override\_kernel\_check=true

1. daemon支持以下选项，用于为容器设置默认的限制，

--storage-opt overlay2.basesize=128M 指定默认限制的大小，若lcrc run时也指定 了--storeage-opt size选项，则以run时指定来生效，若daemon跟lcrc run时都不指定大小，则表示不限制。

1. 需要开启文件系统Project和Project Quota属性。

* 新格式化文件系统并mount

# mkfs.ext4 -O quota,project /dev/sdb   
# mount -o prjquota /dev/sdb /var/lib/lcrd

参数

create/run时指定--storage-opt参数。

| **参数项** | **参数说明** | **取值范围** | **是否必选** |
| --- | --- | --- | --- |
| --storage-opt size=${rootfsSize} | 限制容器rootfs存储空间。 | rootfsSize解析出的大小为int64范围内以字节表示的正数，默认单位为B，也可指定为([kKmMgGtTpP])?[iI]?[bB]?$ | 否 |

示例

在lcrc run/create命令行上通过已有参数“--storage-opt size=<value>”来设置限额。其中value是一个正数，单位可以是[kKmMgGtTpP]?[iI]?[bB]?，在不带单位的时候默认单位是字节。

$ [root@localhost ~]# lcrc run -ti --storage-opt size=10M busybox   
/ # df -h   
Filesystem Size Used Available Use% Mounted on   
overlay 10.0M 48.0K 10.0M 0% /   
none 64.0M 0 64.0M 0% /dev   
none 10.0M 0 10.0M 0% /sys/fs/cgroup   
tmpfs 64.0M 0 64.0M 0% /dev   
shm 64.0M 0 64.0M 0% /dev/shm   
/dev/mapper/vg--data-ext41   
 9.8G 51.5M 9.2G 1% /etc/hostname   
/dev/mapper/vg--data-ext41   
 9.8G 51.5M 9.2G 1% /etc/resolv.conf   
/dev/mapper/vg--data-ext41   
 9.8G 51.5M 9.2G 1% /etc/hosts   
tmpfs 3.9G 0 3.9G 0% /proc/acpi   
tmpfs 64.0M 0 64.0M 0% /proc/kcore   
tmpfs 64.0M 0 64.0M 0% /proc/keys   
tmpfs 64.0M 0 64.0M 0% /proc/timer\_list   
tmpfs 64.0M 0 64.0M 0% /proc/sched\_debug   
tmpfs 3.9G 0 3.9G 0% /proc/scsi   
tmpfs 64.0M 0 64.0M 0% /proc/fdthreshold   
tmpfs 64.0M 0 64.0M 0% /proc/fdenable   
tmpfs 3.9G 0 3.9G 0% /sys/firmware   
/ #   
/ # dd if=/dev/zero of=/home/img bs=1M count=12 && sync   
dm-4: write failed, project block limit reached.   
10+0 records in   
9+0 records out   
10432512 bytes (9.9MB) copied, 0.011782 seconds, 844.4MB/s   
/ # df -h | grep overlay   
overlay 10.0M 10.0M 0 100% /   
/ #

约束

1. 限额只针对rw层。

overlay2的限额是针对容器的rw层的，镜像的大小不计算在内。

1. 内核支持并使能。

内核必须支持ext4的project quota功能，并在mkfs的时候要加上-O quota,project，挂载的时候要加上-o prjquota。任何一个不满足，在使用--storage-opt size=<value>时都将报错。

$ [root@localhost ~]# lcrc run -it --storage-opt size=10Mb busybox df -h   
Error response from daemon: Failed to prepare rootfs with error: time="2019-04-09T05:13:52-04:00" level=fatal msg="error creating read-   
write layer with ID "a4c0e55e82c55e4ee4b0f4ee07f80cc2261cf31b2c2dfd628fa1fb00db97270f": --storage-opt is supported only for overlay over   
xfs or ext4 with 'pquota' mount option"

1. 限制额度的说明。
   1. 限制的额度大于lcrd的root所在分区的size时，在容器内用df看到的文件系统的额度是lcrd的root所在分区的size，而不是设置的限额。
   2. --storage-opt size=0代表不限制，且设置值不能小于4096。size的精度为1个字节，如果指定精度含小数个字节，小数部分被忽略，如指定size=0.1实际等同于size=0不限制。（受计算机存储浮点数精度的限制，即0.999999999999999999999999999与1是等价的，具体的9的个数不同计算机可能存在差异，故设置4095.999999999999999999999999999与4096等价，其它情况类似），注意lcrc inspect显示原始命令行指定形式，如果含小数字节，需自行忽略小数部分。
   3. 限制的额度过小时，比如--storage-opt size=4k，可能会导致容器无法启动，因为启动容器本身需要创建一些文件。
   4. 上一次启动lcrd时，lcrd的root所在分区挂载时加了-o prjquota选项，这次启动时不加，那么上一次启动中创建的带quota的容器的设置值不生效。
   5. daemon端配额--storage-opt overlay2.basesize，其取值范围与--storage-opt size相同。
2. 指定storage-opt为4k时，轻量级容器启动与docker有差异

使用选项 storage-opt size=4k 和镜像 rnd-dockerhub.huawei.com/official/ubuntu-arm64:latest 运行容器。

docker启动失败。

[root@localhost ~]# docker run -itd --storage-opt size=4k rnd-dockerhub.huawei.com/official/ubuntu-arm64:latest   
docker: Error response from daemon: symlink /proc/mounts /var/lib/docker/overlay2/e6e12701db1a488636c881b44109a807e187b8db51a50015db34a131294fcf70-init/merged/etc/mtab: disk quota exceeded.   
See 'docker run --help'.

轻量级容器不报错，正常启动

[root@localhost ~]# lcrc run -itd --storage-opt size=4k rnd-dockerhub.huawei.com/official/ubuntu-arm64:latest   
636480b1fc2cf8ac895f46e77d86439fe2b359a1ff78486ae81c18d089bbd728   
[root@localhost ~]# lcrc ps   
STATUS PID IMAGE COMMAND EXIT\_CODE RESTART\_COUNT STARTAT FINISHAT RUNTIME ID NAMES   
running 17609 rnd-dockerhub.huawei.com/official/ubuntu-arm64:latest /bin/bash 0 0 2 seconds ago - lcr 636480b1fc2c 636480b1fc2cf8ac895f46e77d86439fe2b359a1ff78486ae81c18d089bbd728

在启动容器的过程中，如果需要在容器的rootfs路径下创建文件，若镜像本身占用的大小超过4k，且此时的quota设置为4k，则创建文件必定失败。

docker在启动容器的过程中，会比isulad创建更多的挂载点，用于挂载host上的某些路径到容器中，如/proc/mounts, /dev/shm等，如果镜像内本身不存在这些文件，则会创建，根据上述原因该操作会导致文件创建失败，因而容器启动失败。

轻量级容器在启动容器过程中，使用默认配置时，挂载点较少，如/proc，或/sys等路径不存在时，才会创建。用例中的镜像rnd-dockerhub.huawei.com/official/ubuntu-arm64:latest本身含有/proc， /sys等，因此整个启动容器的过程中，都不会有新文件或路径生成，故轻量级容器启动过程不会报错。为验证这一过程，当把镜像替换为rnd-dockerhub.huawei.com/official/busybox-aarch64:latest时，由于该镜像内无/proc存在，轻量级容器启动一样会报错。

[root@localhost ~]# lcrc run -itd --storage-opt size=4k rnd-dockerhub.huawei.com/official/busybox-aarch64:latest   
8e893ab483310350b8caa3b29eca7cd3c94eae55b48bfc82b350b30b17a0aaf4   
Error response from daemon: Start container error: runtime error: 8e893ab483310350b8caa3b29eca7cd3c94eae55b48bfc82b350b30b17a0aaf4:tools/lxc\_start.c:main:404 starting container process caused "Failed to setup lxc,   
please check the config file."

1. 其他说明。

使用限额功能的lcrd切换数据盘时，需要保证被切换的数据盘使用`prjquota`选项挂载，且/var/lib/lcrd/storage/overlay2目录的挂载方式与/var/lib/lcrd相同。



切换数据盘时需要保证/var/lib/lcrd/storage/overlay2的挂载点被卸载。

### 限制容器内文件句柄数

描述

可以通过参数限制容器中可以打开的文件句柄数。

用法

lcrc create/run时使用--files-limit来限制容器中可以打开的文件句柄数

参数

create/run时指定--files-limit参数。

| **参数项** | **参数说明** | **取值范围** | **是否必选** |
| --- | --- | --- | --- |
| --files-limit | 限制容器中可以打开的文件句柄数。 | 64位整数(int64)。可以为0、负，但不能超过2的63 次方减 1，0、负表示不做限制（max）。  由于创建容器的过程中会临时打开一些句柄，所以此值不能设置的太小，不然容器可能不受files limit的限制（如果设置的数小于当前已经打开的句柄数，会导致cgroup文件写不进去），建议大于30。 | 否 |

示例

在运行容器时，直接加上--files-limit n 即可，如：

lcrc run -ti --files-limit 1024 busybox bash

约束

1. 使用--files-limit参数传入一个很小的值，如1，可能导致容器启动失败。

[root@localhost ~]# lcrc run -itd --files-limit 1 rnd-dockerhub.huawei.com/official/busybox-aarch64   
004858d9f9ef429b624f3d20f8ba12acfbc8a15bb121c4036de4e5745932eff4   
Error response from daemon: Start container error: Container is not running:004858d9f9ef429b624f3d20f8ba12acfbc8a15bb121c4036de4e5745932eff4

而docker会启动成功，其files.limit cgroup值为max。

[root@localhost ~]# docker run -itd --files-limit 1 rnd-dockerhub.huawei.com/official/busybox-aarch64   
ef9694bf4d8e803a1c7de5c17f5d829db409e41a530a245edc2e5367708dbbab   
[root@localhost ~]# docker exec -it ef96 cat /sys/fs/cgroup/files/files.limit   
max

根因是lxc和runc启动过程的原理不一样，lxc创建cgroup子组后先设置files.limit值，再将容器进程的PID写入该子组的cgroup.procs文件，此时该进程已经打开超过1个句柄，因而写入报错导致启动失败。runc创建cgroup子组后先将容器进程的PID写入该子组的cgroup.procs文件，再设置files.limit值，此时由于该子组内的进程已经打开超过1个句柄，因而写入files.limit不会生效，内核也不会报错，容器启动成功。

### 限制容器内可以创建的进程/线程数

描述

可以通过参数限制容器中能够创建的进程/线程数。

用法

在容器create/run时，使用参数--pids-limit来限制容器中可以创建的进程/线程数。

参数

create/run时指定--pids-limit参数。

| **参数项** | **参数说明** | **取值范围** | **是否必选** |
| --- | --- | --- | --- |
| --pids-limit | 限制容器中可以打开的文件句柄数。 | 64位整数(int64)。可以为0、负，但不能超过2的63 次方减 1，0、负表示不做限制（max）。 | 否 |

示例

在运行容器时，直接加上--pids-limit n 即可，如：

lcrc run -ti --pids-limit 1024 busybox bash

约束

由于创建容器的过程中会临时创建一些进程，所以此值不能设置的太小，不然容器可能起不来，建议大于10。

### 配置容器内的ulimit值

描述

可以通过参数控制执行程序的资源。

用法

在容器create/run时配置--ulimit参数，或通过daemon端配置，控制容器中执行程序的资源。

参数

通过两种方法配置ulimit

1. lcrc create/run时使用--ulimit <type>=<soft>[:<hard>]来控制shell执行程序的资源。

| **参数项** | **参数说明** | **取值范围** | **是否必选** |
| --- | --- | --- | --- |
| --ulimit | 限制shell执行程序的资源 | soft/hard是64位整数(int64)。soft取值 <= hard取值，如果仅仅指定了soft的取值，则hard=soft。对于某些类型的资源并不支持负数，详见下表 | 否 |

1. 通过daemon端参数或配置文件

详见4.1 命令行参数说明与配置文件中的--default-ulimits相关选项。

--ulimit可以对以下类型的资源进行限制。

| **类型** | **说明** | **取值范围** |
| --- | --- | --- |
| core | limits the core file size (KB) | 64位整数(INT64)，无单位。可以为0、负、其中-1表示UNLIMITED，即不做限制，其余的负数会被强制转换为一个大的正整数。 |
| cpu | max CPU time (MIN) |
| data | max data size (KB) |
| fsize | maximum filesize (KB) |
| locks | max number of file locks the user can hold |
| memlock | max locked-in-memory address space (KB) |
| msgqueue | max memory used by POSIX message queues (bytes) |
| nice | nice priority |
| nproc | max number of processes |
| rss | max resident set size (KB) |
| rtprio | max realtime priority |
| rttime | realtime timeout |
| sigpending | max number of pending signals |
| stack | max stack size (KB) |
| nofile | max number of open file descriptors | 64位整数(int64)，无单位。不可以为负，负数被强转为大数，设置时会出现Operation not permitted |

示例

在容器的创建或者运行时，加上--ulimit <type>=<soft>[:<hard>]即可，如：

lcrc create/run -tid --ulimit nofile=1024:2048 busybox sh

约束

不能在daemon.json和/etc/sysconfig/iSulad文件（或lcrd命令行）中同时配置ulimit限制，否则lcrd启动会报错。

## 特权容器

### 场景说明

iSulad默认启动的是普通容器，普通容器适合启动普通进程，其权限非常受限，仅具备/etc/default/lcrd/config.json中capabilities所定义的默认权限。当需要特权操作（比如/sys）下的设备时，需要特权容器完成这些操作，使用该特性，容器内的root将拥有宿主机的root权限， 否则，容器内的root在只是宿主机的普通用户权限。

### 使用限制

特权容器为容器提供了所有功能，还解除了设备cgroup控制器强制执行的所有限制，具备以下特性：

* Secomp不block任何系统调用
* /sys、/proc路径可写
* 容器内能访问主机上所有设备
* 系统的权能将全部打开

普通容器默认权能为：

| **Capability Key** | **Capability Description** |
| --- | --- |
| SETPCAP | 修改进程权能 |
| MKNOD | 允许使用mknod()系统调用创建特殊文件 |
| AUDIT\_WRITE | 向内核审计日志写记录 |
| CHOWN | 对文件的 UIDs 和 GIDs 做任意的修改(参考 chown(2)) |
| NET\_RAW | 使用 RAW 和 PACKET sockets；为透明代理绑定任何地址 |
| DAC\_OVERRIDE | 忽略文件的DAC访问限制 |
| FOWNER | 忽略文件属主ID必须和进程用户ID相匹配的限制 |
| FSETID | 允许设置文件的setuid位 |
| KILL | 允许对不属于自己的进程发送信号 |
| SETGID | 允许改变进程的组ID |
| SETUID | 允许改变进程的用户ID |
| NET\_BIND\_SERVICE | 允许绑定到小于1024的端口 |
| SYS\_CHROOT | 允许使用chroot()系统调用 |
| SETFCAP | 允许向其他进程转移能力以及删除其他进程的能力 |

当容器为特权模式时，将添加以下权能

| **Capability Key** | **Capability Description** |
| --- | --- |
| SYS\_MODULE | 加载和卸载内核模块 |
| SYS\_RAWIO | 允许直接访问/devport,/dev/mem,/dev/kmem及原始块设备 |
| SYS\_PACCT | 允许执行进程的BSD式审计 |
| SYS\_ADMIN | 允许执行系统管理任务，如加载或卸载文件系统、设置磁盘配额等 |
| SYS\_NICE | 允许提升优先级及设置其他进程的优先级 |
| SYS\_RESOURCE | 忽略资源限制 |
| SYS\_TIME | 允许改变系统时钟 |
| SYS\_TTY\_CONFIG | 允许配置TTY设备 |
| AUDIT\_CONTROL | 启用和禁用内核审计；修改审计过滤器规则；提取审计状态和过滤规则 |
| MAC\_ADMIN | 覆盖强制访问控制 (Mandatory Access Control (MAC))，为Smack Linux安全模块(Linux Security Module (LSM)) 而实现 |
| MAC\_OVERRIDE | 允许 MAC 配置或状态改变。为 Smack LSM 而实现 |
| NET\_ADMIN | 允许执行网络管理任务 |
| SYSLOG | 执行特权 syslog(2) 操作 |
| DAC\_READ\_SEARCH | 忽略文件读及目录搜索的DAC访问限制 |
| LINUX\_IMMUTABLE | 允许修改文件的IMMUTABLE和APPEND属性标志 |
| NET\_BROADCAST | 允许网络广播和多播访问 |
| IPC\_LOCK | 允许锁定共享内存片段 |
| IPC\_OWNER | 忽略IPC所有权检查 |
| SYS\_PTRACE | 允许跟踪任何进程 |
| SYS\_BOOT | 允许重新启动系统 |
| LEASE | 允许修改文件锁的FL\_LEASE标志 |
| WAKE\_ALARM | 触发将唤醒系统的功能，如设置 CLOCK\_REALTIME\_ALARM 和 CLOCK\_BOOTTIME\_ALARM 定时器 |
| BLOCK\_SUSPEND | 可以阻塞系统挂起的特性 |

### 使用指导

iSulad使用--privileged给容器添加特权模式，在非必要情况下，不要给容器添加特权，遵循最小特权原则，减少存在的安全风险。

lcrc run --rm -it --privileged busybox

## CRI接口

### 描述

CRI API 接口是由kubernetes 推出的容器运行时接口，CRI定义了容器和镜像的服务接口。ISulad使用CRI接口，实现和kubernetes 的对接。

因为容器运行时与镜像的生命周期是彼此隔离的，因此需要定义两个服务。该接口使用[Protocol Buffer](https://developers.google.com/protocol-buffers/)定义，基于[gRPC](https://grpc.io/)。

当前实现CRI版本为v1alpha1版本，官方API描述文件如下：

<https://github.com/kubernetes/kubernetes/blob/release-1.9/pkg/kubelet/apis/cri/v1alpha1/runtime/api.proto>，

ISulad使用的为pass使用的1.9.7版本API描述文件，与官方API略有出入，以本文档描述的接口为准。



CRI接口websocket流式服务，服务端监听地址为127.0.0.1，端口为10251，为固定端口，不支持外部配置。

### 接口

各接口中可能用到的参数清单如下，部分参数暂不支持配置，已在配置中标出。

接口参数列表

* **DNSConfig**

配置sandbox的DNS服务器和搜索域

| **参数成员** | **描述** |
| --- | --- |
| repeated string servers | 集群的DNS服务器列表 |
| repeated string searches | 集群的DNS搜索域列表 |
| repeated string options | DNS可选项列表，参考<https://linux.die.net/man/5/resolv.conf> |

* **Protocol**

协议的enum值列表

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| TCP = 0↵ | TCP协议 |
| UDP = 1 | UDP协议 |

* **PortMapping**

指定sandbox的端口映射配置

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| [Protocol](#d0e4578) protocol | 端口映射使用的协议 |
| int32 container\_port | 容器内的端口号 |
| int32 host\_port | 主机上的端口号 |
| string host\_ip | 主机IP地址 |

* **MountPropagation**

挂载传播属性的enum列表

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| PROPAGATION\_PRIVATE = 0 | 无挂载传播属性，即linux中的private |
| PROPAGATION\_HOST\_TO\_CONTAINER = 1 | 挂载属性能从host传播到容器中，即linux中的rslave |
| PROPAGATION\_BIDIRECTIONAL = 2 | 挂载属性能在host和容器中双向传播，即linux中的rshared |

* **Mount**

Mount指定host上的一个挂载卷挂载到容器中（只支持文件和文件夹)

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string container\_path | 容器中的路径 |
| string host\_path | 主机上的路径 |
| bool readonly | 是否配置在容器中是只读的， 默认值： false |
| bool selinux\_relabel | 是否设置SELinux标签（不支持配置） |
| [MountPropagation](#zh-cn_topic_0182207110_li201899371871) propagation | 挂载传播属性配置（取值**0/1/2**，分别对应**private/rslave/rshared**传播属性） **默认值：0** |

* **NamespaceOption**

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| bool host\_network | 是否使用host的网络命名空间 |
| bool host\_pid | 是否使用host的PID命名空间 |
| bool host\_ipc | 是否使用host的IPC命名空间 |

* **Capability**

包含待添加与待删除的权能信息

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| repeated string add\_capabilities | 待新增的权能 |
| repeated string drop\_capabilities | 待删除的权能 |

* **Int64Value**

int64类型的封装

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| int64 value | 实际的int64值 |

* **UInt64Value**

uint64类型的封装

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| uint64 value | 实际的uint64值 |

* **LinuxSandboxSecurityContext**

配置sandbox的linux安全选项。

注意，这些安全选项不会应用到sandbox中的容器中，也可能不适用于没有任何running进程的sandbox

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| [NamespaceOption](#zh-cn_topic_0182207110_li1182444614213) namespace\_options | 配置sandbox的命名空间选项 |
| [SELinuxOption](#zh-cn_topic_0182207110_li816815620237) selinux\_options | 配置SELinux选项（不支持） |
| [Int64Value](#zh-cn_topic_0182207110_li1439643415372) run\_as\_user | 配置sandbox中进程的uid |
| bool readonly\_rootfs | 配置sandbox的根文件系统是否只读 |
| repeated int64 supplemental\_groups | 配置除主GID之外的sandbox的1号进程用户组信息 |
| bool privileged | 配置sandbox是否为特权容器 |
| string seccomp\_profile\_path | seccomp配置文件路径，有效值为：  // unconfined: 不配置seccomp  // localhost/<配置文件的全路径>： 安装在系统上的配置文件路径  // <配置文件的全路径>： 配置文件全路径  // 默认不配置，即unconfined。 |

* **LinuxPodSandboxConfig**

设定和Linux主机及容器相关的一些配置

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string cgroup\_parent | sandbox的cgroup父路径，runtime可根据实际情况使用cgroupfs或systemd的语法。（不支持配置） |
| [LinuxSandboxSecurityContext](#d0e4913) security\_context | sandbox的安全属性 |
| map<string, string> sysctls | sandbox的linux sysctls配置 |

* **PodSandboxMetadata**

Sandbox元数据包含构建sandbox名称的所有信息，鼓励容器运行时在用户界面中公开这些元数据以获得更好的用户体验，例如，运行时可以根据元数据生成sandbox的唯一命名。

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string name | sandbox的名称 |
| string uid | sandbox的UID |
| string namespace | sandbox的命名空间 |
| uint32 attempt | 尝试创建sandbox的次数，默认为0 |

* **PodSandboxConfig**

包含创建sandbox的所有必选和可选配置信息

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| [PodSandboxMetadata](#zh-cn_topic_0182207110_li2359918134912) metadata | sandbox的元数据，这项信息唯一标识一个sandbox，runtime必须利用此信息确保操作正确，runtime也可以根据此信息来改善用户体验，例如构建可读的sandbox名称。 |
| string hostname | sandbox的hostname |
| string log\_directory | 配置sandbox内的容器的日志文件所存储的文件夹 |
| [DNSConfig](#zh-cn_topic_0182207110_li1197211439388) dns\_config | sandbox的DNS配置 |
| repeated [PortMapping](#zh-cn_topic_0182207110_li01684174215) port\_mappings | sandbox的端口映射 |
| map<string, string> labels | 可用于标识单个或一系列sandbox的键值对 |
| map<string, string> annotations | 存储任意信息的键值对，这些值是不可更改的，且能够利用PodSandboxStatus接口查询 |
| [LinuxPodSandboxConfig](#d0e5000) linux | 与linux主机相关的可选项 |

* **PodSandboxNetworkStatus**

描述sandbox的网络状态

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string ip | sandbox的ip地址 |
| string name | sandbox内的网络接口名 |
| string network | 附加网络的名称 |

* **Namespace**

命名空间选项

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| [NamespaceOption](#zh-cn_topic_0182207110_li1182444614213) options | Linux 命名空间选项 |

* **LinuxPodSandboxStatus**

描述Linux sandbox的状态

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| [Namespace](#zh-cn_topic_0182207110_li523062951815) **namespaces** | sandbox命名空间 |

* **PodSandboxState**

sandbox状态值的enum数据

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| SANDBOX\_READY = 0 | sandbox处于ready状态 |
| SANDBOX\_NOTREADY = 1 | sandbox处于非ready状态 |

* **PodSandboxStatus**

描述Podsandbox的状态信息

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string id | sandbox的ID |
| [PodSandboxMetadata](#zh-cn_topic_0182207110_li2359918134912) metadata | sandbox的元数据 |
| [PodSandboxState](#zh-cn_topic_0182207110_li1818214574195) state | sandbox的状态值 |
| int64 created\_at | sandbox的创建时间戳，单位纳秒 |
| repeated [PodSandboxNetworkStatus](#zh-cn_topic_0182207110_li255017717184) networks | sandbox的多平面网络状态 |
| [LinuxPodSandboxStatus](#zh-cn_topic_0182207110_li313112151212) linux | Linux规范的sandbox状态 |
| map<string, string> labels | 可用于标识单个或一系列sandbox的键值对 |
| map<string, string> annotations | 存储任意信息的键值对，这些值是不可被runtime更改的 |

* **PodSandboxStateValue**

对[PodSandboxState](#zh-cn_topic_0182207110_li1818214574195)的封装

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| [PodSandboxState](#zh-cn_topic_0182207110_li1818214574195) state | sandbox的状态值 |

* **PodSandboxFilter**

用于列出sandbox时添加过滤条件，多个条件取交集显示

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string id | sandbox的ID |
| [PodSandboxStateValue](#zh-cn_topic_0182207110_li64922552019) state | sandbox的状态 |
| map<string, string> label\_selector | sandbox的labels，label只支持完全匹配，不支持正则匹配 |

* **PodSandbox**

包含最小化描述一个sandbox的数据

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string id | sandbox的ID |
| [PodSandboxMetadata](#zh-cn_topic_0182207110_li2359918134912) metadata | sandbox的元数据 |
| [PodSandboxState](#zh-cn_topic_0182207110_li1818214574195) state | sandbox的状态值 |
| int64 created\_at | sandbox的创建时间戳，单位纳秒 |
| map<string, string> labels | 可用于标识单个或一系列sandbox的键值对 |
| map<string, string> annotations | 存储任意信息的键值对，这些值是不可被runtime更改的 |

* **KeyValue**

键值对的封装

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string key | 键 |
| string value | 值 |

* **SELinuxOption**

应用于容器的SELinux标签

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string user | 用户 |
| string role | 角色 |
| string type | 类型 |
| string level | 级别 |

* **ContainerMetadata**

Container元数据包含构建container名称的所有信息，鼓励容器运行时在用户界面中公开这些元数据以获得更好的用户体验，例如，运行时可以根据元数据生成container的唯一命名。

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string name | container的名称 |
| uint32 attempt | 尝试创建container的次数，默认为0 |

* **ContainerState**

容器状态值的enum列表

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| CONTAINER\_CREATED = 0 | container创建完成 |
| CONTAINER\_RUNNING = 1 | container处于运行状态 |
| CONTAINER\_EXITED = 2 | container处于退出状态 |
| CONTAINER\_UNKNOWN = 3 | 未知的容器状态 |

* **ContainerStateValue**

封装[ContainerState](#zh-cn_topic_0182207110_li65182518309)的数据结构

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| [ContainerState](#zh-cn_topic_0182207110_li65182518309) **state** | 容器状态值 |

* **ContainerFilter**

用于列出container时添加过滤条件，多个条件取交集显示

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string id | container的ID |
| [PodSandboxStateValue](#zh-cn_topic_0182207110_li64922552019) state | container的状态 |
| string pod\_sandbox\_id | sandbox的ID |
| map<string, string> label\_selector | container的labels，label只支持完全匹配，不支持正则匹配 |

* **LinuxContainerSecurityContext**

指定应用于容器的安全配置

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [Capability](#zh-cn_topic_0182207110_li115631277434) capabilities | 新增或去除的权能 |
| bool privileged | 指定容器是否未特权模式， **默认值：false** |
| [NamespaceOption](#zh-cn_topic_0182207110_li1182444614213) namespace\_options | 指定容器的namespace选项 |
| [SELinuxOption](#zh-cn_topic_0182207110_li816815620237) selinux\_options | SELinux context(可选配置项) **未使用** |
| [Int64Value](#zh-cn_topic_0182207110_li1439643415372) run\_as\_user | 运行容器进程的UID。 一次只能指定run\_as\_user与run\_as\_username其中之一，run\_as\_username优先生效 |
| string run\_as\_username | 运行容器进程的用户名。 如果指定，用户必须存在于容器映像中（即在映像内的/etc/passwd中），并由运行时在那里解析; 否则，运行时必须出错 |
| bool readonly\_rootfs | 设置容器中根文件系统是否为只读 **默认值由config.json配置** |
| repeated int64 supplemental\_groups | 容器运行的除主GID外首进程组的列表 |
| string apparmor\_profile | 容器的AppArmor配置文件 **未使用** |
| string seccomp\_profile\_path | 容器的seccomp配置文件路径 |
| bool no\_new\_privs | 是否在容器上设置no\_new\_privs的标志 |

* **LinuxContainerResources**

指定Linux容器资源的特定配置

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| int64 cpu\_period | CPU CFS（完全公平调度程序）周期。 **默认值：0** |
| int64 cpu\_quota | CPU CFS（完全公平调度程序）配额。 **默认值：0** |
| int64 cpu\_shares | 所占CPU份额（相对于其他容器的相对权重）。 **默认值：0** |
| int64 memory\_limit\_in\_bytes | 内存限制（字节）。 **默认值：0** |
| int64 oom\_score\_adj | OOMScoreAdj用于调整oom-killer。 **默认值：0** |
| string cpuset\_cpus | 指定容器使用的CPU核心。 **默认值：“”** |
| string cpuset\_mems | 指定容器使用的内存节点。 **默认值：“”** |

* **Image**

Image信息描述一个镜像的基本数据。

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string id | 镜像ID |
| repeated string repo\_tags | 镜像tag 名称 repo\_tags |
| repeated string repo\_digests | 镜像digest信息 |
| uint64 size | 镜像大小 |
| [Int64Value](#zh-cn_topic_0182207110_li1439643415372) uid | 镜像默认用户UID |
| string username | 镜像默认用户名称 |

* **ImageSpec**

表示镜像的内部数据结构，当前，ImageSpec只封装容器镜像名称

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string image | 容器镜像名 |

* **StorageIdentifier**

唯一定义storage的标识

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string uuid | 设备的UUID |

* **FilesystemUsage**

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| int64 timestamp | 收集文件系统信息时的时间戳 |
| [StorageIdentifier](#zh-cn_topic_0182207110_li3285401546) storage\_id | 存储镜像的文件系统UUID |
| [UInt64Value](#zh-cn_topic_0182207110_li1886455713453) used\_bytes | 存储镜像元数据的大小 |
| [UInt64Value](#zh-cn_topic_0182207110_li1886455713453) inodes\_used | 存储镜像元数据的inodes个数 |

* **AuthConfig**

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string username | 下载镜像使用的用户名 |
| string password | 下载镜像使用的密码 |
| string auth | 下载镜像时使用的认证信息，base64编码 |
| string server\_address | 下载镜像的服务器地址，暂不支持 |
| string identity\_token | 用于与镜像仓库鉴权的令牌信息，暂不支持 |
| string registry\_token | 用于与镜像仓库交互的令牌信息，暂不支持 |

* **Container**

用于描述容器信息，例如ID, 状态等。

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string id | container的ID |
| string pod\_sandbox\_id | 该容器所属的sandbox的ID |
| [ContainerMetadata](#d0e5645) metadata | container的元数据 |
| [ImageSpec](#zh-cn_topic_0182207110_li597891416252) image | 镜像规格 |
| string image\_ref | 代表容器使用的镜像，对大多数runtime来产，这是一个image ID值 |
| [ContainerState](#zh-cn_topic_0182207110_li65182518309) state | container的状态 |
| int64 created\_at | container的创建时间戳，单位为纳秒 |
| map<string, string> labels | 可用于标识单个或一系列container的键值对 |
| map<string, string> annotations | 存储任意信息的键值对，这些值是不可被runtime更改的 |

* **ContainerStatus**

用于描述容器状态信息

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string id | container的ID |
| [ContainerMetadata](#d0e5645) metadata | container的元数据 |
| [ContainerState](#zh-cn_topic_0182207110_li65182518309) state | container的状态 |
| int64 created\_at | container的创建时间戳，单位为纳秒 |
| int64 started\_at | container启动时的时间戳，单位为纳秒 |
| int64 finished\_at | container退出时的时间戳，单位为纳秒 |
| int32 exit\_code | 容器退出码 |
| [ImageSpec](#zh-cn_topic_0182207110_li597891416252) image | 镜像规格 |
| string image\_ref | 代表容器使用的镜像，对大多数runtime来产，这是一个image ID值 |
| string reason | 简要描述为什么容器处于当前状态 |
| string message | 易于人工阅读的信息，用于表述容器处于当前状态的原因 |
| map<string, string> labels | 可用于标识单个或一系列container的键值对 |
| map<string, string> annotations | 存储任意信息的键值对，这些值是不可被runtime更改的 |
| repeated [Mount](#zh-cn_topic_0182207110_li6779341144216) mounts | 容器的挂载点信息 |
| string log\_path | 容器日志文件路径，该文件位于[PodSandboxConfig](#zh-cn_topic_0182207110_li253629701)中配置的log\_directory文件夹下 |

* **ContainerStatsFilter**

用于列出container stats时添加过滤条件，多个条件取交集显示

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string id | container的ID |
| string pod\_sandbox\_id | sandbox的ID |
| map<string, string> label\_selector | container的labels，label只支持完全匹配，不支持正则匹配 |

* **ContainerStats**

用于列出container stats时添加过滤条件，多个条件取交集显示

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| [ContainerAttributes](#zh-cn_topic_0182207110_li6207185712312)attributes | 容器的信息 |
| [CpuUsage](#zh-cn_topic_0182207110_li1367131122711)cpu | CPU使用情况 |
| [MemoryUsage](#d0e6663)memory | 内存使用情况 |
| [FilesystemUsage](#zh-cn_topic_0182207110_li1606183118189)writable\_layer | 可写层使用情况 |

* **ContainerAttributes**

列出container的基本信息

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| string id | 容器的ID |
| [ContainerMetadata](#d0e5645)metadata | 容器的metadata |
| map<string,string> labels | 可用于标识单个或一系列container的键值对 |
| map<string,string> annotations | 存储任意信息的键值对，这些值是不可被runtime更改的 |

* **CpuUsage**

列出container的CPU使用信息

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| int64 timestamp | 时间戳 |
| UInt64Value usage\_core\_nano\_seconds | CPU的使用值，单位/纳秒 |

* **MemoryUsage**

列出container的内存使用信息

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| int64 timestamp | 时间戳 |
| UInt64Value working\_set\_bytes | 内存的使用值 |

* **FilesystemUsage**

列出container的读写层信息

| **参数成员**↵ | **描述**↵ |
| --- | --- |
| int64 timestamp | 时间戳 |
| StorageIdentifier storage\_id | 可写层目录 |
| UInt64Value used\_bytes | 镜像在可写层的占用字节 |
| UInt64Value inodes\_used | 镜像在可写层的占用inode数 |

* **Device**

指定待挂载至容器的主机卷

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string container\_path | 容器内的挂载路径 |
| string host\_path | 主机上的挂载路径 |
| string permissions | 设备的Cgroup权限，（r允许容器从指定的设备读取; w允许容器从指定的设备写入; m允许容器创建尚不存在的设备文件) |

* **LinuxContainerConfig**

包含特定于Linux平台的配置

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [LinuxContainerResources](#zh-cn_topic_0182207110_li2050214613477) resources | 容器的资源规范 |
| [LinuxContainerSecurityContext](#d0e5819) security\_context | 容器的Linux容器安全配置 |

* **ContainerConfig**

包含用于创建容器的所有必需和可选字段

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [ContainerMetadata](#d0e5645) metadata | 容器的元数据。 此信息将唯一标识容器，运行时应利用此信息来确保正确操作。 运行时也可以使用此信息来提升UX（用户体检设计），例如通过构造可读名称。**(必选)** |
| [ImageSpec](#zh-cn_topic_0182207110_li597891416252) image | 容器使用的镜像 **(必选)** |
| repeated string command | 待执行的命令 **默认值： "/bin/sh"** |
| repeated string args | 待执行命令的参数 |
| string working\_dir | 命令执行的当前工作路径 |
| repeated [KeyValue](#d0e5559) envs | 在容器中配置的环境变量 |
| repeated [Mount](#zh-cn_topic_0182207110_li6779341144216) mounts | 待在容器中挂载的挂载点信息 |
| repeated [Device](#d0e6749) devices | 待在容器中映射的设备信息 |
| map<string, string> labels | 可用于索引和选择单个资源的键值对。 |
| map<string, string> annotations | 可用于存储和检索任意元数据的非结构化键值映射。 |
| string log\_path | 相对于PodSandboxConfig.LogDirectory的路径，用于存储容器主机上的日志（STDOUT和STDERR）。 |
| bool stdin | 是否打开容器的stdin |
| bool stdin\_once | 当某次连接stdin的数据流断开时，是否立即断开其他与stdin连接的数据流（**暂不支持**） |
| bool tty | 是否使用伪终端连接容器的stdio |
| [LinuxContainerConfig](#d0e6789) linux | linux系统上容器的特定配置信息 |

* **NetworkConfig**

Runtime的网络配置

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string pod\_cidr | Pod IP 地址使用的CIDR |

* **RuntimeConfig**

Runtime的网络配置

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [NetworkConfig](#zh-cn_topic_0182207110_li12222146464) network\_config | Runtime的网络配置 |

* **RuntimeCondition**

描述runtime的状态信息

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string type | Runtime状态的类型 |
| bool status | Runtime状态 |
| string reason | 简要描述runtime状态变化的原因 |
| string message | 具备可阅读性的信息表明runtime状态变化的原因 |

* **RuntimeStatus**

Runtime的状态

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| repeated RuntimeCondition conditions | 描述当前runtime状态的列表 |

#### Runtime服务

Runtime服务中包含操作pod和容器的接口，以及查询runtime自身配置和状态信息的接口。

##### RunPodSandbox

接口原型

rpc RunPodSandbox(RunPodSandboxRequest) returns (RunPodSandboxResponse) {}

接口描述

创建和启动一个pod sandbox，若运行成功，sandbox处于ready状态。

注意事项

1. 启动sandbox的默认镜像为rnd-dockerhub.huawei.com/library/pause-${machine}:3.0， 其中${machine}为架构，在x86\_64上，machine的值为amd64，在arm64上，machine的值为aarch64，当前rnd-dockerhub仓库上只有amd64和aarch64镜像可供下载，若机器上无此镜像，请确保机器能从rnd-dockerhub下载，若要使用其它镜像，请参考[iSulad部署配置](#_ZH-CN_TOPIC_0184808042)中的pod-sandbox-image指定镜像。
2. 由于容器命名以[PodSandboxMetadata](#zh-cn_topic_0182207110_li2359918134912)中的字段为来源，且以下划线"\_"为分割字符，因此限制metadata中的数据不能包含下划线，否则会出现sandbox运行成功，但无法使用[ListPodSandbox](#_ZH-CN_TOPIC_0184808098)接口查询的现象。

参数：RunPodSandboxRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [PodSandboxConfig](#zh-cn_topic_0182207110_li253629701) config | sandbox的配置，具体参数含义参考左侧链接。 |

返回值：RunPodSandboxResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| string pod\_sandbox\_id | 成功，返回response数据，具体参数参考左侧链接 |

##### StopPodSandbox

接口原型

rpc StopPodSandbox(StopPodSandboxRequest) returns (StopPodSandboxResponse) {}

接口描述

停止pod sandbox，停止sandbox容器，回收分配给sandbox的网络资源（比如IP地址）。如果有任何running的容器属于该sandbox，则必须被强制停止。

注意事项

参数: StopPodSandboxRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string pod\_sandbox\_id | sandbox的id |

返回值: StopPodSandboxResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| 无 | 无 |

##### RemovePodSandbox

接口原型

rpc RemovePodSandbox(RemovePodSandboxRequest) returns (RemovePodSandboxResponse) {}

接口描述

删除sandbox，如果有任何running的容器属于该sandbox，则必须被强制停止和删除，如果sandbox已经被删除，不能返回错误。

注意事项

1. 删除sandbox时，不会删除sandbox的网络资源，在删除pod前必须先调用StopPodSandbox才能清理网络资源，调用者应当保证在删除sandbox之前至少调用一次StopPodSandbox。

参数: RemovePodSandboxRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string pod\_sandbox\_id | sandbox的id |

返回值: RemovePodSandboxResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| 无 | 无 |

##### PodSandboxStatus

接口原型

rpc PodSandboxStatus(PodSandboxStatusRequest) returns (PodSandboxStatusResponse) {}

接口描述

查询sandbox的状态，如果sandbox不存在，返回错误。

注意事项

参数: PodSandboxStatusRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string pod\_sandbox\_id | sandbox的id |
| bool verbose | 标识是否显示sandbox的一些额外信息。（暂不支持配置） |

返回值: PodSandboxStatusResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| [PodSandboxStatus](#zh-cn_topic_0182207110_li146986172010) status | sandbox的状态信息 |
| map<string, string> info | sandbox的额外信息，key是任意string，value是json格式的字符串，这些信息可以是任意调试内容。当verbose为true时info不能为空。（暂不支持配置） |

##### ListPodSandbox

接口原型

rpc ListPodSandbox(ListPodSandboxRequest) returns (ListPodSandboxResponse) {}

接口描述

返回sandbox信息的列表，支持条件过滤。

注意事项

参数: ListPodSandboxRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [PodSandboxFilter](#zh-cn_topic_0182207110_li17913177201) filter | 条件过滤参数，具体内容参考左侧链接 |

返回值: ListPodSandboxResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| repeated [PodSandbox](#d0e5491) items | sandbox信息的列表 |

##### CreateContainer

grpc::Status CreateContainer(grpc::ServerContext \*context, const runtime::CreateContainerRequest \*request, runtime::CreateContainerResponse \*reply) {}

接口描述

在PodSandbox内创建一个容器

注意事项

1. 请求CreateContainerRequest 中的sandbox\_config与传递给RunPodSandboxRequest以创建PodSandbox的配置相同。 它再次传递，只是为了方便参考。 PodSandboxConfig是不可变的，在pod的整个生命周期内保持不变。

2. 由于容器命名以[ContainerMetadata](#d0e5645)中的字段为来源，且以下划线"\_"为分割字符，因此限制metadata中的数据不能包含下划线，否则会出现sandbox运行成功，但无法使用[ListContainers](#_ZH-CN_TOPIC_0184808103)接口查询的现象。

参数： CreateContainerRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string pod\_sandbox\_id | 待在其中创建容器的PodSandbox的ID。 |
| [ContainerConfig](#zh-cn_topic_0182207110_li9517163811284) config | 容器的配置信息 |
| [PodSandboxConfig](#zh-cn_topic_0182207110_li253629701) sandbox\_config | PodSandbox的配置信息 |

补充 ：annotations

可用于存储和检索任意元数据的非结构化键值映射。有一些字段由于cri接口没有提供特定的参数，可通过该字段将参数传入

* 自定义

|  |  |
| --- | --- |
| **自定义 key:value** | **描述**↵ |
| cgroup.pids.max:int64\_t | 用于限制容器内的进/线程数（set -1 for unlimited） |

返回值 ：CreateContainerResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| string container\_id | 创建完成的容器ID |

##### StartContainer

接口原型

rpc StartContainer(StartContainerRequest) returns (StartContainerResponse) {}

接口描述

启动一个容器。

注意事项

参数： StartContainerRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string container\_id | 容器id |

返回值： StartContainerResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| 无 | 无 |

##### StopContainer

接口原型

rpc StopContainer(StopContainerRequest) returns (StopContainerResponse) {}

接口描述

停止一个running的容器，支持配置优雅停止时间timeout，如果容器已经停止，不能返回错误。

注意事项

参数： StopContainerRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string container\_id | 容器id |
| int64 timeout | 强制停止容器前的等待时间，默认值为0，即强制停止容器。 |

返回值: StopContainerResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| 无 | 无 |

##### RemoveContainer

接口原型

rpc RemoveContainer(RemoveContainerRequest) returns (RemoveContainerResponse) {}

接口描述

删除一个容器，如果容器正在运行，必须强制停止，如果容器已经被删除，不能返回错误。

注意事项

参数: RemoveContainerRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string container\_id | 容器id |

返回值: RemoveContainerResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| 无 | 无 |

##### ListContainers

接口原型

rpc ListContainers(ListContainersRequest) returns (ListContainersResponse) {}

接口描述

返回container信息的列表，支持条件过滤。

注意事项

参数： ListContainersRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [ContainerFilter](#zh-cn_topic_0182207110_li780212262306) filter | 条件过滤参数，具体内容参考左侧链接 |

返回值： ListContainersResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| repeated [Container](#zh-cn_topic_0182207110_li2063672883012) containers | 容器信息的列表 |

##### ContainerStatus

接口原型

rpc ContainerStatus(ContainerStatusRequest) returns (ContainerStatusResponse) {}

接口描述

返回容器状态信息，如果容器不存在，则返回错误。

注意事项

参数: ContainerStatusRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string container\_id | 容器id |
| bool verbose | 标识是否显示sandbox的一些额外信息。（暂不支持配置） |

返回值: ContainerStatusResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| [ContainerStatus](#zh-cn_topic_0182207110_li1234063113301) status | 容器的状态信息 |
| map<string, string> info | sandbox的额外信息，key是任意string，value是json格式的字符串，这些信息可以是任意调试内容。当verbose为true时info不能为空。（暂不支持配置） |

##### UpdateContainerResources

接口原型

rpc UpdateContainerResources(UpdateContainerResourcesRequest) returns (UpdateContainerResourcesResponse) {}

接口描述

该接口用于更新容器资源配置。

注意事项

1. 该接口仅用于更新容器的资源配置，不能用于更新Pod的资源配置。

2. 当前不支持更新容器oom\_score\_adj配置

参数: UpdateContainerResourcesRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string container\_id | 容器id |
| [LinuxContainerResources](#zh-cn_topic_0182207110_li2050214613477) linux | linux资源配置信息 |

返回值: UpdateContainerResourcesResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| 无 | 无 |

##### ExecSync

接口原型

rpc ExecSync(ExecSyncRequest) returns (ExecSyncResponse) {}

接口描述

以同步的方式在容器中执行命令，采用的gRPC通讯方式。

注意事项

执行执行一条单独的命令，不能打开终端与容器交互。

参数：ExecSyncRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string container\_id | 容器ID |
| repeated string cmd | 待执行命令 |
| int64 timeout | 停止命令的超时时间（秒）。 默认值：0（无超时限制）。 **未使用** |

返回值: ExecSyncResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| bytes stdout | 捕获命令标准输出 |
| bytes stderr | 捕获命令标准错误输出 |
| int32 exit\_code | 退出代码命令完成。 默认值：0（成功）。 |

##### Exec

接口原型

rpc Exec(ExecRequest) returns (ExecResponse) {}

接口描述

在容器中执行命令，采用的gRPC通讯方式从CRI服务端获取url，再通过获得的url与websocket服务端建立长连接，实现与容器的交互。

注意事项

执行执行一条单独的命令，也能打开终端与容器交互。stdin/stdout/stderr之一必须是真的。如果tty为真，stderr必须是假的。 不支持多路复用, 在这种情况下, stdout和stderr的输出将合并为单流。

参数: ExecRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string container\_id | 容器ID |
| repeated string cmd | 待执行的命令 |
| bool tty | 是否在TTY中执行命令 |
| bool stdin | 是否流式标准输入 |
| bool stdout | 是否流式标准输出 |
| bool stderr | 是否流式输出标准错误 |

返回值: ExecResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| string url | exec流服务器的完全限定URL |

##### Attach

接口原型

rpc Attach(AttachRequest) returns (AttachResponse) {}

接口描述

接管容器的1号进程，采用的gRPC通讯方式从CRI服务端获取url，再通过获得的url与websocket服务端建立长连接，实现与容器的交互。

参数: AttachRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string container\_id | 容器ID |
| bool tty | 是否在TTY中执行命令 |
| bool stdin | 是否流式标准输入 |
| bool stdout | 是否流式标准输出 |
| bool stderr | 是否流式输出标准错误 |

返回值: ExecResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| string url | attach流服务器的完全限定URL |

##### PortForward

暂不支持

##### ContainerStats

接口原型

rpc ContainerStats(ContainerStatsRequest) returns (ContainerStatsResponse) {}

接口描述

返回单个容器占用资源信息

参数： ContainerStatsRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| string container\_id | 容器id |

返回值： ContainerStatsResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| [ContainerStats](#zh-cn_topic_0182207110_li55689514215) stats | 容器信息，具体内容参考左侧链接。注：disk和inodes只支持oci格式镜像起的容器查询 |

##### ListContainerStats

接口原型

rpc ListContainerStats(ListContainerStatsRequest) returns (ListContainerStatsResponse) {}

接口描述

返回多个容器占用资源信息，支持条件过滤

参数： ListContainerStatsRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [ContainerStatsFilter](#zh-cn_topic_0182207110_li285981611148) filter | 条件过滤参数，具体内容参考左侧链接 |

返回值： ListContainerStatsResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| repeated [ContainerStats](#zh-cn_topic_0182207110_li55689514215) stats | 容器信息的列表,具体内容参考左侧链接。注：disk和inodes只支持oci格式镜像起的容器查询 |

##### UpdateRuntimeConfig

接口原型

rpc UpdateRuntimeConfig(UpdateRuntimeConfigRequest) returns (UpdateRuntimeConfigResponse);

接口描述

提供标准的CRI接口，目的为了更新网络插件的Pod CIDR，当前CNI网络插件无需更新Pod CIDR，因此该接口只会记录访问日志。

注意事项

接口操作不会对系统管理信息修改，只是记录一条日志。

参数： UpdateRuntimeConfigRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [RuntimeConfig](#zh-cn_topic_0182207110_li544075884710) runtime\_config | 包含Runtime要配置的信息 |

返回值： UpdateRuntimeConfigResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| 无 | 无 |

##### Status

接口原型

rpc Status(StatusRequest) returns (StatusResponse) {};

接口描述

获取runtime和pod的网络状态，在获取网络状态时，会触发网络配置的刷新。

注意事项

如果网络配置刷新失败，不会影响原有配置；只有刷新成功时，才会覆盖原有配置。

参数： StatusRequest

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| bool verbose | 是否显示关于Runtime额外的信息（暂不支持） |

返回值： StatusResponse

|  |  |
| --- | --- |
| **返回值** | **描述** |
| [RuntimeStatus](#zh-cn_topic_0182207110_li15257663554) status | Runtime的状态 |
| map<string, string> info | Runtime额外的信息，info的key为任意值，value为json格式，可包含任何debug信息；只有Verbose为true是才应该被赋值 |

#### Image服务

提供了从镜像仓库拉取、查看、和移除镜像的gRPC API。

##### ListImages

接口原型

rpc ListImages(ListImagesRequest) returns (ListImagesResponse) {}

接口描述

列出当前已存在的镜像信息。

注意事项

为统一接口，对于embedded格式镜像，可以通过cri images查询到。但是因embedded镜像不是标准OCI镜像，因此查询得到的结果有以下限制：

1. 因embedded镜像无镜像ID，显示的镜像ID为 镜像的config digest。
2. 因embedded镜像本身无digest仅有config的digest，且格式不符合OCI镜像规范，因此无法显示digest。

参数

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [ImageSpec](#zh-cn_topic_0182207110_li597891416252) filter | 筛选的镜像名称 |

返回值

|  |  |
| --- | --- |
| **返回值** | **描述** |
| repeated [Image](#zh-cn_topic_0182207110_li597891416252) images | 镜像信息列表 |

##### ImageStatus

接口原型

rpc ImageStatus(ImageStatusRequest) returns (ImageStatusResponse) {}

接口描述

查询指定镜像信息。

注意事项

1. 查询指定镜像信息，若镜像不存在，则返回ImageStatusResponse，其中Image设置为nil。
2. 为统一接口，对于embedded格式镜像，因不符合OCI格式镜像，缺少字段，无法通过本接口进行查询。

参数

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [ImageSpec](#zh-cn_topic_0182207110_li597891416252) image | 镜像名称 |
| bool verbose | 查询额外信息，暂不支持，无额外信息返回 |

返回值

|  |  |
| --- | --- |
| **返回值** | **描述** |
| [Image](#zh-cn_topic_0182207110_li597891416252) image | 镜像信息 |
| map<string, string> info | 镜像额外信息，暂不支持，无额外信息返回 |

##### PullImage

接口原型

rpc PullImage(PullImageRequest) returns (PullImageResponse) {}

接口描述

下载镜像。

注意事项

当前支持下载public镜像，使用用户名、密码、auth信息下载私有镜像，不支持authconfig中的server\_address、identity\_token、registry\_token字段。

参数

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [ImageSpec](#zh-cn_topic_0182207110_li597891416252) image | 要下载的镜像名称 |
| [AuthConfig](#zh-cn_topic_0182207110_li1017394413316) auth | 下载私有镜像时的验证信息 |
| [PodSandboxConfig](#zh-cn_topic_0182207110_li253629701) sandbox\_config | 在Pod上下文中下载镜像（暂不支持） |

返回值

|  |  |
| --- | --- |
| **返回值** | **描述** |
| string image\_ref | 返回已下载镜像信息 |

##### RemoveImage

接口原型

rpc RemoveImage(RemoveImageRequest) returns (RemoveImageResponse) {}

接口描述

删除指定镜像。

注意事项

为统一接口，对于embedded格式镜像，因不符合OCI格式镜像，缺少字段，无法通过本接口使用image id进行删除。

参数

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| [ImageSpec](#zh-cn_topic_0182207110_li597891416252) image | 要删除的镜像名称或者ID |

返回值

|  |  |
| --- | --- |
| **返回值** | **描述** |
| 无 | 无 |

##### ImageFsInfo

接口原型

rpc ImageFsInfo(ImageFsInfoRequest) returns (ImageFsInfoResponse) {}

接口描述

查询存储镜像的文件系统信息。

注意事项

查询到的为镜像元数据下的文件系统信息。

参数

|  |  |
| --- | --- |
| **参数成员** | **描述** |
| 无 | 无 |

返回值

|  |  |
| --- | --- |
| **返回值** | **描述** |
| repeated [FilesystemUsage](#zh-cn_topic_0182207110_li1606183118189) image\_filesystems | 镜像存储文件系统信息 |

### 约束

1. 如果创建sandbox时，PodSandboxConfig参数中配置了log\_directory，则所有属于该sandbox的container在创建时必须在ContainerConfig中指定log\_path，否则可能导致容器无法使用CRI接口启动，甚至无法使用CRI接口删除。

容器的真实LOGPATH=log\_directory/log\_path，如果log\_path不配置，那么最终的LOGPATH会变为LOGPATH=log\_directory。

* 如果该路径不存在，isulad在启动容器时会创建一个软链接，指向最终的容器日志真实路径，此时log\_directory变成一个软链接，此时有两种情况：
  + 1. 第一种情况，如果该sandbox里其它容器也没配置log\_path，在启动其它容器时，log\_directory会被删除，然后重新指向新启动容器的log\_path，导致之前启动的容器日志指向后面启动容器的日志。
    2. 第二种情况，如果该sandbox里其它容器配置了log\_path，则该容器的LOGPATH=log\_directory/log\_path，由于log\_directory实际是个软链接，使用log\_directory/log\_path为软链接指向容器真实日志路径时，创建会失败。
* 如果该路径存在，isulad在启动容器时首先会尝试删除该路径（非递归），如果该路径是个文件夹，且里面有内容，删除会失败，从而导致创建软链接失败，容器启动失败，删除该容器时，也会出现同样的现象，导致删除失败。

1. 如果创建sandbox时，PodSandboxConfig参数中配置了log\_directory，且container创建时在ContainerConfig中指定log\_path，那么最终的LOGPATH=log\_directory/log\_path，isulad不会递归的创建LOGPATH，因而用户必须保证dirname(LOGPATH)存在，即最终的日志文件的上一级路径存在。
2. 如果创建sandbox时，PodSandboxConfig参数中配置了log\_directory，如果有两个或多个container创建时在ContainerConfig中指定了同一个log\_path，或者不同的sandbox内的容器最终指向的LOGPATH是同一路径，若容器启动成功，则后启动的容器日志路径会覆盖掉之前启动的容器日志路径。
3. 如果远程镜像仓库中镜像内容发生变化，调用CRI Pull image接口重新下载该镜像时，若本地原来存储有原镜像，则原镜像的镜像名称、TAG会变更为“none”

举例如下：

本地已存储镜像：

IMAGE TAG IMAGE ID SIZE   
rnd-dockerhub.huawei.com/pproxyisulad/test latest 99e59f495ffaa 753kB

远程仓库中rnd-dockerhub.huawei.com/pproxyisulad/test:latest 镜像更新后，重新下载后：

IMAGE TAG IMAGE ID SIZE   
<none> <none> 99e59f495ffaa 753kB   
rnd-dockerhub.huawei.com/pproxyisulad/test latest d8233ab899d41 1.42MB

使用lcrc images 命令行查询，REF显示为"-"：

REF IMAGE ID CREATED SIZE   
rnd-dockerhub.huawei.com/pproxyisulad/test:latest d8233ab899d41 2019-02-14 19:19:37 1.42MB   
- 99e59f495ffaa 2016-05-04 02:26:41 753kB

## 镜像管理

### docker镜像管理

#### 从镜像仓库拉取镜像

描述

从镜像仓库拉取镜像到本地。

描述

lcrc pull [OPTIONS] NAME[:TAG|@DIGEST]

参数

login命令支持参数请参见“附录 >表4-3”。

示例

$ lcrc pull localhost:5000/official/busybox   
Image "localhost:5000/official/busybox" pulling   
Image "localhost:5000/official/busybox@sha256:bf510723d2cd2d4e3f5ce7e93bf1e52c8fd76831995ac3bd3f90ecc866643aff" pulled

#### 删除镜像

描述

删除一个或多个镜像。

用法

lcrc rmi [OPTIONS] IMAGE [IMAGE...]

参数

login命令支持参数请参见“附录 >表4-4”。

示例

$ lcrc rmi rnd-dockerhub.huawei.com/official/busybox   
Image "rnd-dockerhub.huawei.com/official/busybox" removed

#### 加载镜像

描述

从一个tar包加载镜像。该tar包必须是使用docker save命令导出的tar包或格式一致的tar包。

用法

lcrc load [OPTIONS]

参数

login命令支持参数请参见“附录 >表4-5”。

示例

$ lcrc load -i busybox.tar   
Load image from "/root/busybox.tar" success

#### 列出镜像

描述

列出当前环境中所有镜像。

用法

lcrc images

参数

login命令支持参数请参见“附录 >表4-6”。

示例

$ lcrc images   
REF IMAGE ID CREATED SIZE   
rnd-dockerhub.huawei.com/official/busybox:latest e4db68de4ff2 2019-06-15 08:19:54 1.376 MB

#### 检视镜像

描述

返回该镜像的配置信息。可以使用-f参数过滤出需要的信息。

用法

lcrc inspect [options] CONTAINER|IMAGE [CONTAINER|IMAGE...]

参数

login命令支持参数请参见“附录 >表4-7”。

示例

$ lcrc inspect -f "{{json .image.id}}" rnd-dockerhub.huawei.com/official/busybox   
"e4db68de4ff27c2adfea0c54bbb73a61a42f5b667c326de4d7d5b19ab71c6a3b"

#### 双向认证

描述

开启该功能后isulad和镜像仓库之间的通信采用https通信，isulad和镜像仓库都会验证对方的合法性。

用法

要支持该功能，需要镜像仓库支持该功能，同时isulad也需要做相应的配置：

1、修改isulad的配置(默认路径/etc/isulad/daemon.json)，将配置里的use-decrypted-key项配置为false。

2、需要将相关的证书放置到/etc/isulad/certs.d目录下对应的镜像仓库命名的文件夹下，证书具体的生成方法见docker的官方链接：

https://docs.docker.com/engine/security/certificates/

https://docs.docker.com/engine/security/https/

3、执行systemctl restart lcrd重启isulad

参数

可以在/etc/isulad/daemon.json中配置参数，也可以在启动isulad时携带参数：

lcrd --use-decrypted-key=false

示例

配置use-decrypted-key参数为false

$ cat /etc/isulad/daemon.json   
{   
 "group": "lcrd",   
 "graph": "/var/lib/lcrd",   
 "state": "/var/run/lcrd",   
 "engine": "lcr",   
 "log-level": "ERROR",   
 "pidfile": "/var/run/lcrd.pid",   
 "log-opts": {   
 "log-file-mode": "0600",   
 "log-path": "/var/lib/lcrd",   
 "max-file": "1",   
 "max-size": "30KB"   
 },   
 "log-driver": "stdout",   
 "hook-spec": "/etc/default/lcrd/hooks/default.json",   
 "start-timeout": "2m",   
 "storage-driver": "overlay2",   
 "storage-opts": [   
 "overlay2.override\_kernel\_check=true"   
 ],   
 "registry-mirrors": [   
 "docker.io"   
 ],   
 "insecure-registries": [   
 "rnd-dockerhub.huawei.com"   
 ],   
 "pod-sandbox-image": "",   
 "image-opt-timeout": "5m",   
 "native.umask": "secure",   
 "network-plugin": "",   
 "cni-bin-dir": "",   
 "cni-conf-dir": "",   
 "image-layer-check": false,   
 "use-decrypted-key": false,   
 "insecure-skip-verify-enforce": false   
}

将证书放到对应的目录下

$ pwd   
/etc/isulad/certs.d/my.csp-edge.com:5000   
$ ls   
ca.crt tls.cert tls.key

重启isulad

$ systemctl restart lcrd

执行pull命令从仓库下载镜像

$ lcrc pull my.csp-edge.com:5000/ubuntu   
Image "my.csp-edge.com:5000/ubuntu" pulling   
Image "my.csp-edge.com:5000/ubuntu@sha256:f1bdc62115dbfe8f54e52e19795ee34b4473babdeb9bc4f83045d85c7b2ad5c0" pulled

### embedded镜像管理

#### 加载镜像

描述

根据embedded镜像的manifest加载镜像。注意--type的值必须填写embedded。

用法

lcrc load [OPTIONS] --input=FILE --type=TYPE

参数

login命令支持参数请参见“附录 >表4-5”。

示例

$ lcrc load -i test.manifest --type embedded   
Load image from "/root/work/bugfix/tmp/ci\_testcase\_data/embedded/img/test.manifest" success

#### 列出镜像

描述

列出当前环境中所有镜像。

用法

lcrc images [OPTIONS]

参数

login命令支持参数请参见“附录 >表4-6”。

示例

$ lcrc images   
REF IMAGE ID CREATED SIZE   
test:v1 9319da1f5233 2018-03-01 10:55:44 1.273 MB

#### 检视镜像

描述

返回该镜像的配置信息。可以使用-f参数过滤出需要的信息。

用法

lcrc inspect [options] CONTAINER|IMAGE [CONTAINER|IMAGE...]

参数

login命令支持参数请参见“附录 >表4-7”。

示例

$ lcrc inspect -f "{{json .created}}" test:v1   
"2018-03-01T15:55:44.322987811Z"

#### 删除镜像

描述

删除一个或多个镜像。

用法

lcrc rmi [OPTIONS] IMAGE [IMAGE...]

参数

login命令支持参数请参见“附录 > 表4-4”。

示例

$ lcrc rmi test:v1   
Image "test:v1" removed

## 容器健康状态检查

### 场景说明

在实际的生产环境中，开发者提供的应用程序或者平台提供的服务难免存在bug，因此，一套管理系统对运行的应用程序进行周期性的健康检查和修复就是不可或缺的。容器健康检查机制便添加了用户定义的对容器进行健康检查的功能。在容器创建时配置--health-cmd选项，在容器内部周期性地执行命令，通过命令的返回值来监测容器的健康状态。

### 配置方法

在容器启动时的配置：

lcrc run -itd --health-cmd "echo iSulad >> /tmp/health\_check\_file || exit 1" --health-interval 5m --health-timeout 3s --health-exit-on-unhealthy centos bash

可配置的选项：

* --health-cmd，必选，在容器内执行的命令。返回值为0表示成功，非0表示失败。
* --health-interval，默认 30s，最大为int64上限（纳秒），自定义配置最小值1s，相邻两次命令执行的间隔时间（注：入参0s时视为default）。
* --health-timeout，默认 30s，最大为int64上限（纳秒），自定义配置最小值1s，单次检查命令执行的时间上限，超时则任务命令执行失败（注：入参0s时视为default）
* --health-start-period，默认 0s，最大为int64上限（纳秒），自定义配置最小值1s，容器初始化时间。
* --health-retries，默认 3，最大为int32上限，健康检查失败最大的重试次数。
* --health-exit-on-unhealthy，默认false，检测到容器非健康时是否杀死容器。

### 检查规则

1. 容器启动后，容器状态中显示health:starting。
2. 经过start-period时间后开始，以interval为间隔周期性在容器中执行CMD。即：当一次命令执行完毕后，经过interval时间，执行下一次命令。
3. 若CMD命令在timeout限制的时间内执行完毕，并且返回值为0，则视为一次检查成功。否则视为一次检查失败。检查成功后，容器状态变为health:healthy。
4. 若CMD命令连续retries次检查失败，则容器状态变为health:unhealthy。失败后容器也会继续进行健康检查。
5. 容器状态为health:unhealthy时，任意一次检查成功会使得容器状态变为health:healthy。
6. 设置--exit-on-unhealthy的情况下，如果容器因为非被杀死退出（退出返回值137）后，健康检查只有容器在重新启动后才会继续生效。
7. CMD执行完毕或超时时，docker daemon会将这次检查的起始时间、返回值和标准输出记录到容器的配置文件中。最多记录5条。此外，容器的配置文件中还存储着健康检查的相关参数。
8. 运行中的容器的健康检查状态也会被写入容器配置中。通过lcrc inspect可以看到。

"Health": {   
 "Status": "healthy",   
 "FailingStreak": 0,   
 "Log": [   
 {   
 "Start": "2018-03-07T07:44:15.481414707-05:00",   
 "End": "2018-03-07T07:44:15.556908311-05:00",   
 "ExitCode": 0,   
 "Output": ""   
 },   
 {   
 "Start": "2018-03-07T07:44:18.557297462-05:00",   
 "End": "2018-03-07T07:44:18.63035891-05:00",   
 "ExitCode": 0,   
 "Output": ""   
 },   
 ......   
}

### 使用限制

* 容器内健康检查的状态信息最多保存5条。会保存最后得到的5条记录。
* 容器启动时若健康检查相关参数配置为0，则按照默认值处理。
* 带有健康检查配置的容器启动后，若iSulad daemon退出，则健康检查不会执行。iSulad daemon再次启动后，正在运行且带有健康检查配置的容器其健康状态会变为starting。之后检查规则同上。
* 如果健康检查从第一次开始便一直失败，其状态保持与之前一致（starting）,直到达到指定失败次数（--health-retries）后变为unhealthy，或者检查成功后变为healthy。

## 查询信息

### 查询服务版本信息

描述

lcrc version 命令用于查询iSulad服务的版本信息。

用法

lcrc version

实例

查询版本信息

lcrc version

如果lcrd服务正常运行，则可以查看到客户端、服务端以及OCI config的版本等信息。

Client:   
 Version: 1.0.31   
 Git commit: fa7f9902738e8b3d7f2eb22768b9a1372ddd1199   
 Built: 2019-07-30T04:21:48.521198248-04:00   
   
Server:   
 Version: 1.0.31   
 Git commit: fa7f9902738e8b3d7f2eb22768b9a1372ddd1199   
 Built: 2019-07-30T04:21:48.521198248-04:00   
   
OCI config:   
 Version: 1.0.0-rc5-dev   
 Default file: /etc/default/lcrd/config.json

若lcrd服务未运行，则仅仅查询到客户端的信息，并提示无法连接到服务端。

Client:   
 Version: 1.0.31   
 Git commit: fa7f9902738e8b3d7f2eb22768b9a1372ddd1199   
 Built: 2019-07-30T04:21:48.521198248-04:00   
   
Can not connect with server.Is the iSulad daemon running on the host?

因此，lcrc version命令也常常用来检验lcrd是否正常运行。

### 查询系统级信息

描述

lcrc info命令用于对系统级信息，以及容器和镜像数目等信息的查询。

用法

lcrc info

示例

查询系统级信息，可以展示容器数目，镜像数目，内核版本、操作系统等信息

$ lcrc info   
Containers: 2   
 Running: 0   
 Paused: 0   
 Stopped: 2   
Images: 8   
Server Version: 1.0.31   
Logging Driver: json-file   
Cgroup Driverr: cgroupfs   
Hugetlb Pagesize: 2MB   
Kernel Version: 4.19   
Operating System: Fedora 29 (Twenty Nine)   
OSType: Linux   
Architecture: x86\_64   
CPUs: 8   
Total Memory: 7 GB   
Name: localhost.localdomain   
iSulad Root Dir: /var/lib/lcrd

## 安全特性

### seccomp安全配置场景

#### 场景说明

seccomp（**secure computing** **mode**）是linux kernel从2.6.23版本开始引入的一种简洁的sandboxing机制。在一些特定场景下，用户需要在容器中执行一些“特权”操作，但又不想启动特权容器，用户经常会在run时添加--cap-add来获得一些“小范围”的权限。对于安全要求比较严格的容器实例，上述的CAP粒度不一定能够满足安全需要，可使用一些办法精细化控制权限范围。

* 举例

普通容器场景中，用户使用-v将宿主机某目录（包含某普通用户无法执行的二进制），映射到容器中。

在容器中，可以将二进制修改权限chmod 4777加入S标志位。这样在宿主机上，原先不能运行二进制的普通用户（或者运行此二进制受限），可以在S标志位的添加动作后，在运行此二进制的时候，获取到二进制自身的权限（比如root），从而提权运行或者访问其他文件。

这个场景，如果在严格安全要求下，需要使用seccomp裁剪chmod、fchmod、fchmodat系统调用。

#### 使用限制

* seccomp可能会影响性能，设置seccomp之前需要对场景进行评估，确定必要时加入。

#### 使用指导

通过--security-opt将配置文件传给要过滤系统调用的容器。

lcrc run -itd --security-opt seccomp=/path/to/seccomp/profile.json rnd-dockerhub.huawei.com/official/busybox



1. 创建容器时通过--security-opt将配置文件传给容器时，采用默认配置文件（/etc/isulad/seccomp\_default.json）

2. 创建容器时--security-opt设置为unconfined时，对容器不过滤系统调用

3. “/path/to/seccomp/profile.json”需要是绝对路径。

获取普通容器的默认seccomp配置

* 启动一个普通容器（或者是带--cap-add的容器），并查看默认权限配置：

cat /etc/isulad/seccomp\_default.json | python -m json.tool > profile.json

可以看到"seccomp"字段中，有很多的"syscalls"，在此基础上，仅提取syscalls的部分，参考定制seccomp配置文件，进行定制化操作。

"defaultAction": "SCMP\_ACT\_ERRNO",   
"syscalls": [   
{   
"action": "SCMP\_ACT\_ALLOW",   
"name": "accept"   
},   
{   
"action": "SCMP\_ACT\_ALLOW",   
"name": "accept4"   
},   
{   
"action": "SCMP\_ACT\_ALLOW",   
"name": "access"   
},   
{   
"action": "SCMP\_ACT\_ALLOW",   
"name": "alarm"   
},   
{   
"action": "SCMP\_ACT\_ALLOW",   
"name": "bind"   
},   
]...

* 查看转换为lxc可识别的seccomp配置

cat /var/lib/lcrd/engines/lcr/74353e38021c29314188e29ba8c1830a4677ffe5c4decda77a1e0853ec8197cd/seccomp

...   
waitpid allow   
write allow   
writev allow   
ptrace allow   
personality allow [0,0,SCMP\_CMP\_EQ,0]   
personality allow [0,8,SCMP\_CMP\_EQ,0]   
personality allow [0,131072,SCMP\_CMP\_EQ,0]   
personality allow [0,131080,SCMP\_CMP\_EQ,0]   
personality allow [0,4294967295,SCMP\_CMP\_EQ,0]   
...

定制seccomp配置文件

在启动容器的时候使用--security-opt引入seccomp配置文件，容器实例会按照配置文件规则进行限制系统API的运行。首先按照获取普通容器的默认seccomp配置获取到全集模板，然后按照本节定制配置文件，启动容器：

lcrc run --rm -it --security-opt seccomp:/path/to/seccomp/profile.json rnd-dockerhub.huawei.com/official/busybox

配置文件模板：

{   
"defaultAction": "SCMP\_ACT\_ALLOW",   
"syscalls": [   
{   
"name": "syscall-name",   
"action": "SCMP\_ACT\_ERRNO",   
"args": null   
}   
]   
}



* defaultAction、syscalls：对应的action的类型是一样的，但其值是不能一样的，目的就是让所有的syscal都有一个默认的action，并且如果syscalls数组中有明确的定义，就以syscalls中的为准，由于defaultAction、action的值不一样，就能保证action不会有冲突。当前支持的action有：

"SCMP\_ACT\_ERRNO"：禁止，并打印错误信息。

"SCMP\_ACT\_ALLOW"：允许。

* syscalls： 数组，可以只有一个syscall，也可以有多个，可以带args，也可以不带。
* name：要过滤的syscall。
* args：数组，里面的每个object的定义如下：

type Arg struct {   
Index uint `json:"index"` //参数的序号，如open(fd, buf, len),fd 对应的就是0，buf为1   
Value uint64 `json:"value"` //跟参数进行比较的值   
ValueTwo uint64 `json:"value\_two"` //仅当Op=MaskEqualTo时起作用，用户传入值跟Value按位与操作后，跟ValueTwo进行比较，若相等则执行action。   
Op Operator `json:"op"`   
}

args中的Op，其取值可以下页面的任意一种：

"SCMP\_CMP\_NE": NotEqualTo

"SCMP\_CMP\_LT": LessThan

"SCMP\_CMP\_LE": LessThanOrEqualTo

"SCMP\_CMP\_EQ": EqualTo

"SCMP\_CMP\_GE": GreaterThanOrEqualTo

"SCMP\_CMP\_GT": GreaterThan

"SCMP\_CMP\_MASKED\_EQ": MaskEqualTo

### capabilities安全配置场景

#### 场景说明

capabilities机制是linux kernel 2.2之后引入的安全特性，用更小的粒度控制超级管理员权限,可以避免使用 root 权限，将root用户的权限细分为不同的领域，可以分别启用或禁用。capabilities详细信息可通过Linux Programmer's Manual进行查看（[capabilities(7) - Linux man page](http://man7.org/linux/man-pages/man7/capabilities.7.html)）：

man capabilities

#### 使用限制

* lcrd默认Capabilities（白名单）配置如下，普通容器进程将默认携带：

"CAP\_CHOWN",   
"CAP\_DAC\_OVERRIDE",   
"CAP\_FSETID",   
"CAP\_FOWNER",   
"CAP\_MKNOD",   
"CAP\_NET\_RAW",   
"CAP\_SETGID",   
"CAP\_SETUID",   
"CAP\_SETFCAP",   
"CAP\_SETPCAP",   
"CAP\_NET\_BIND\_SERVICE",   
"CAP\_SYS\_CHROOT",   
"CAP\_KILL",   
"CAP\_AUDIT\_WRITE"

* 默认的权能配置，包含了CAP\_SETUID和CAP\_FSETID，如host和容器共享目录，容器可对共享目录的二进制文件进行+s为设置，host上的普通用户可使用其进行提权。CAP\_AUDIT\_WRITE，容器可以对host写入，存在一定的风险，如果使用场景不需要，推荐在启动容器的时候使用--cap-drop将其删除。
* 增加Capabilities意味着容器进程具备更大的能力，同时也会开发更多的系统调用接口，自然具备更强的破坏能力。

#### 使用指导

iSulad使用--cap-add/--cap-drop给容器增加/删去特定的权限，在非必要情况下，不要给容器增加额外的权限，推荐将容器默认但非必要的权限也去掉。

lcrc run --rm -it --cap-add all --cap-drop SYS\_ADMIN rnd-dockerhub.huawei.com/official/busybox

## 支持OCI hooks

### 描述

支持在容器的生命周期中，运行OCI标准hooks。包括三种类型的hooks：

* prestart hook：在执行lcrc start命令之后，而在容器的1号进程启动之前，被执行。
* poststart hook：在容器1号进程启动之后，而在lcrc start命令返回之前，被执行。
* poststop hook：在容器被停止之后，但是在停止命令返回之前，被执行。

OCI hooks的配置格式规范如下：

* path：格式是字符串，必须项，必须为绝对路径，指定的文件必须有可执行权限。
* args：格式是字符串数组，可选项，语义和execv的args一致。
* env：格式是字符串数组，可选项，语义和环境变量一致，内容为键值对，如："PATH=/usr/bin"。
* timeout：格式是整数，可选项，必须大于0，表示钩子执行的超时时间。如果钩子进程运行时间超过配置的时间，那么钩子进程会被杀死。

hook的配置为json格式，一般存放在json结尾的文件中，示例如下：

{   
 "prestart": [   
 {   
 "path": "/usr/bin/echo",   
 "args": ["arg1", "arg2"],   
 "env": [ "key1=value1"],   
 "timeout": 30   
 },   
 {   
 "path": "/usr/bin/ls",   
 "args": ["/tmp"]   
 }   
 ],   
 "poststart": [   
 {   
 "path": "/usr/bin/ls",   
 "args": ["/tmp"],   
 "timeout": 5   
 }   
 ],   
 "poststop": [   
 {   
 "path": "/tmp/cleanup.sh",   
 "args": ["cleanup.sh", "-f"]   
 }   
 ]   
}

### 接口

lcrd和lcrc都提供了hook的接口，lcrd提供了默认的hook配置，会作用于所有容器；而lcrc提供的hook接口，只会作用于当前创建的容器。

lcrd提供的默认OCI hooks接口：

* 通过/etc/isulad/daemon.json配置文件的hook-spec配置项设置hook配置的文件路径："hook-spec": "/etc/default/lcrd/hooks/default.json"
* 通过lcrd --hook-spec参数设置hook配置的文件路径

lcrc提供的OCI hooks接口：

* lcrc create --hook-spec：指定hook配置的json文件的路径
* lcrc run --hook-spec：指定hook配置的json文件的路径

run的配置其实也是在create阶段生效了。

### 使用限制

* hook-spec指定的路径必须是绝对路径；
* hook-spec指定的文件必须存在；
* hook-spec指定的路径对应的必须是普通文本文件，格式为json；
* hook-spec指定的文件大小不能超过10MB；
* hooks配置的path必须为绝对路径；
* hooks配置的path指定文件必须存在；
* hooks配置的path指定文件必须有可执行权限；
* hooks配置的path指定文件的owner必须是root；
* hooks配置的path指定文件必须只有root有写权限；
* hooks配置的timeout必须大于0。

# 附录

[4.1 命令行参数说明](#_ZH-CN_TOPIC_0189976936)

[4.2 CNI配置参数](#_ZH-CN_TOPIC_0184808159)

## 命令行参数说明

login命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **login** | -H, --host | 指定要连接的LCRD socket文件路径 |
| -p, --password | 登录镜像仓库的密码 |
| --password-stdin | 从标准输入获取仓库的密码 |
| -u, --username | 登录镜像仓库的用户名 |

logout命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **logout** | -H, --host | 指定要连接的LCRD socket文件路径 |

pull命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **pull** | -H, --host | 指定要连接的LCRD socket文件路径 |

rmi命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **rmi** | -H, --host | 指定要连接的LCRD socket文件路径 |
| -f, --force | 强制移除镜像 |

load命令参数列表

|  |  |
| --- | --- |
| **命令** | **说明** |
| **load** | 指定要连接的LCRD socket文件路径 |
| 从配置文件上输入 |
| 不使用默认的镜像名称，而是使用TAG指定的名称，type为docker类型时支持该参数 |
| 镜像类型，取值为embedded或docker（默认值） |

images命令参数列表

|  |  |  |
| --- | --- | --- |
| **命令** | **参数** | **说明** |
| **images** | -H, --host | 指定要连接的LCRD socket文件路径 |
| -q, --quit | 只显示镜像名字 |

inspect命令参数列表

| **命令** | **参数** | **说明** |
| --- | --- | --- |
| **inspect** | -H, --host | 指定要连接的LCRD socket文件路径 |
| -f, --format | 使用模板格式化输出 |
| -t, --time | 超时时间的秒数，若在该时间内inspect查询容器信息未执行成功，则停止等待并立即报错，默认为120秒，当配置小于等于0的值，表示不启用timeout机制inspect查询容器信息会一直等待，直到获取容器信息成功后返回。 |

## CNI配置参数

CNI单网络配置参数

| **参数** | **类型** | **是否可选** | **说明** |
| --- | --- | --- | --- |
| cniVersion | string | 必选 | CNI版本号，当前只支持0.3.0，0.3.1。 |
| name | string | 必选 | 网络名称，由用户自定义，需保证唯一。 |
| type | string | 必选 | 网络类型。目前支持的网络类型：  underlay\_ipvlan  overlay\_l2  underlay\_l2  vpc-router  dpdk-direct  phy-direct |
| ipmasp | bool | 可选 | 设置IP masquerade |
| ipam | 结构体 | 可选 | 详细定义参考IPAM参数定义 |
| ipam.type | string | 可选 | IPAM类型，目前支持的类型：  （1）underlay\_l2、overlay\_l2、vpc-router组网默认值distributed\_l2，且只支持distributed\_l2。  （2）underlay\_ipvlan组网，默认distributed\_l2。**CCN场景只支持null、fixed；CCE和FST 5G core场景只支持null、distributed\_l2。**  （3）phy-direct、dpdk-direct组网，默认l2，可选null、distributed\_l2。**FST 5G core场景只支持null、distributed\_l2。**  说明：  超出选择范围（比如host-local），Canal会自动设置为默认，不会返回错误。  null：不使用canal管理ip。  fixed：固定ip，CCN场景使用。  l2：目前没有场景使用。  distributed\_l2：使用分布式小子网管理ip。 |
| ipam.subnet | string | 可选 | 子网信息。Canal支持的subnet mask范围为[8,29]，并且要求IP地址不能为Multicast地址（如224.0.0.0/4），保留地址（240.0.0.0/4），本地link地址（169.254.0.0/16）以及本地loop地址（127.0.0.0/8）。 |
| ipam.gateway | string | 可选 | 网关IP |
| ipam.range-start | string | 可选 | 可用的起始IP地址 |
| ipam.range-end | string | 可选 | 可用的结束IP地址 |
| ipam.routes | 结构体 | 可选 | subnet列表，每个元素都是一个route字典。参考route定义. |
| ipam.routes.dst | string | 可选 | 表示目的网络 |
| ipam.routes.gw | string | 可选 | 表示网关地址 |
| dns | 结构体 | 可选 | 包含一些DNS的特殊值。 |
| dns.nameservers | []string | 可选 | nameservers |
| dns.domain | string | 可选 | domain |
| dns.search | []string | 可选 | search |
| dns.options | []string | 可选 | 选项 |
| multi\_entry | int | 可选 | 表示一个vnic需要的ip数量,范围0~16。对于物理直通，单个网卡最多可申请128个IP。 |
| backup\_mode | bool | 可选 | 表示主备模式，仅用于phy-direct和dpdk-direct组网。 |
| vlanID | int | 可选 | 0~4095,允许PaaS直接指定。 |
| vlan\_inside | bool | 可选 | true表示vlan功能由Node内部实现，false表示vlan在外部实现。 |
| vxlanID | int | 可选 | 0~16777215,允许PaaS直接指定。 |
| vxlan\_inside | bool | 可选 | true表示vlan功能由Node内部实现，false表示vlan在外部实现。 |
| action | string | 可选 | 该参数只能和特殊containerID “000000000000”一起使用。  Create表示创建网络。  Delete表示删除网络。 |
| args | map[string]interface{} | 可选 | 主要描述键值对类型。表4-9 |
| runtimeConfig | 结构体 | 可选 |  |
| capabilities | 结构体 | 可选 |  |

CNI args参数表

| **参数** | **类型** | **是否可选** | **说明** |
| --- | --- | --- | --- |
| K8S\_POD\_NAME | string | 可选 | 申请固定IP（runtimeConfig.ican\_caps.fixed\_ip为true）时需要设置K8S\_POD\_NAME |
| K8S\_POD\_NAMESPACE | string | 可选 | 申请固定IP（runtimeConfig.ican\_caps.fixed\_ip为true）时需要设置K8S\_POD\_NAMESPACE |
| SECURE\_CONTAINER | string | 可选 | 安全容器标志 |
| multi\_port | int | 可选 | 默认值为1，取值范围1-8。只支持phy-direct和dpdk-direct两种类型网络，指定直通网卡数量 |
| phy-direct | string | 可选 | 用于在创建硬直通容器网络时指定接入的网卡 |
| dpdk-direct | string | 可选 | 用于在创建dpdk直通容器网络时指定接入的网卡 |
| tenant\_id | string | 可选 | 租户的ID。  **只支持vpc-router类型网络。** |
| vpc\_id | string | 可选 | VPC的ID。  **只支持vpc-router类型网络。** |
| secret\_name | string | 可选 | 表示k8s apiserver中保存有ak sk的对象名。  **只支持vpc-router类型网络**  参考配置VPC-Router逻辑网络 |
| IP | string | 可选 | 用户指定ip地址，格式“192.168.0.10” |
| K8S\_POD\_NETWORK\_ARGS | string | 可选 | 指定ip地址，格式“192.168.0.10”。若args中IP和K8S\_POD\_NETWORK\_ARGS都不为空，以K8S\_POD\_NETWORK\_ARGS为准。 |
| INSTANCE\_NAME | string | 可选 | INSTANCE ID。  参考支持容器固定IP |
| dist\_gateway\_disable | bool | 可选 | true表示不创建gateway，false表示创建gateway。 |
| phynet | string或[]string | 可选 | 所需加入的的物理平面信息，为预先定义好的物理网络名称，与SNC体系中的呼应，输入两个平面名时，支持主备平面。例如："phy\_net1" 或 ["phy\_net2","phy\_net3"] |
| endpoint\_policies | struct | 可选 | "endpoint\_policies": [  {  "Type": "",  "ExceptionList": [  ""  ],  "NeedEncap": true,  "DestinationPrefix": ""  }  ] |
| port\_map | struct | 可选 | NAT类型网络中，支持容器端口发布至主机端口。  "port\_map": [  {  "local\_port": number,  "host\_port": number,  "protocol": [string…]  }...  ] |

CNI多网络配置参数

| 参数 | 类型 | 是否可选 | 说明 |
| --- | --- | --- | --- |
| cniVersion | string | 必选 | CNI版本号，当前只支持0.3.0，0.3.1。 |
| name | string | 必选 | 网络名称，由用户自定义，需保证唯一。 |
| plugins | struct | 必选 | 具体配置请参见表4-8。 |