# Linux Containers 简介

2015/9/6
renyl

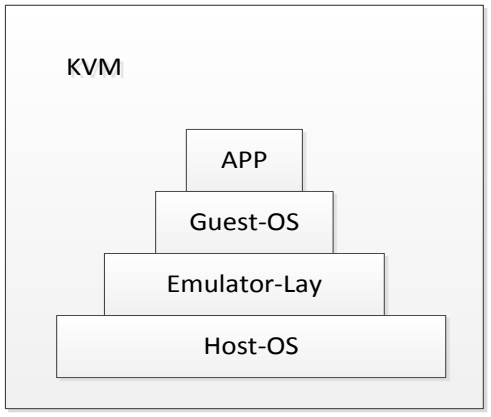
## 1 Linux Containers 介绍

Linux Containers 是一种操作系统级（内核轻量级）虚拟化技术 ，相对传统虚拟化技术 KVM，具有如下特点：

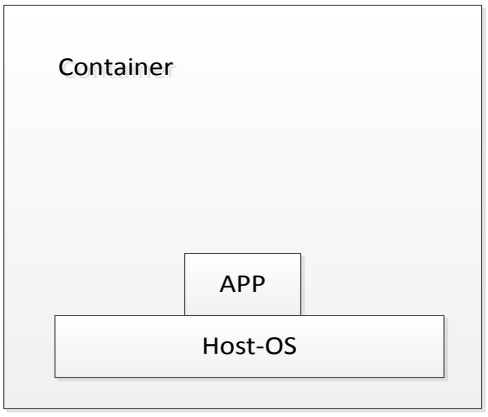
1) Container 与 Host 共用一个内核。
2) Container 相当于一个轻量级 APP sandbox。
3) Container 作为一个普通进程在 Host 上运行。
4) Container 不需要 Host 的 CPU 支持虚拟化。

Container 和 KVM 的系统架构图如下所示：

KVM 系统结构




Container 系统结构




由 Container 和 KVM 的系统结构，可看出 Container 相对于 KVM 具有如下优缺点：

| ID | ITEM | Container | KVM |
|---|---|---|---|
| 1 | Performance | Great | Normal |
| 2 | OS Support | Linux Only | No Limit |
| 3 | Security | Normal | Great |
| 4 | Completeness | Low | High |
| 5 | Complexity | Low | High |

注：本文在如下平台下进行 Linux Containers 研究及测试。

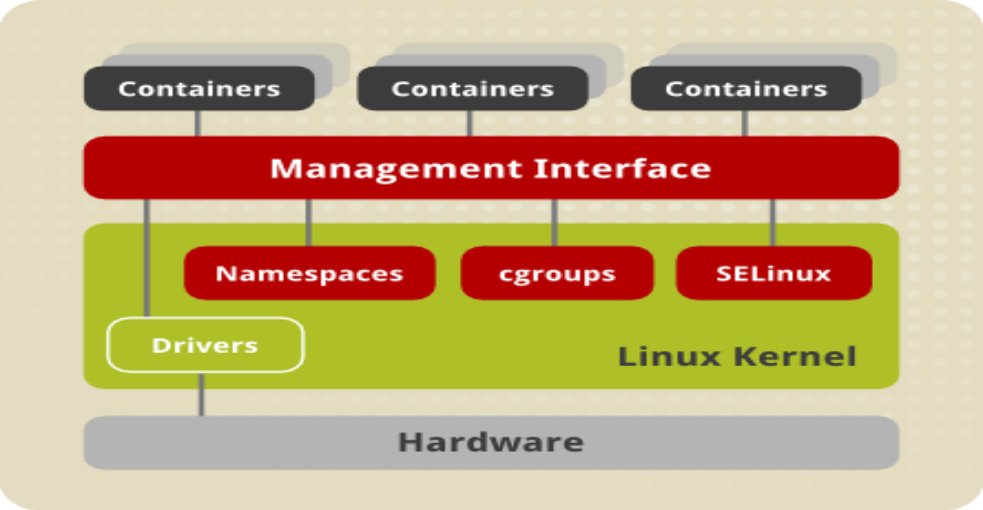
| - | 描述 |
|---|---|
| os | RHEL7.0_x86_64 |
| kernel | kernel-3.10.0-110.el7.x86_64 |
| cpu | Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz |
| glibc | glibc-2.17-52.el7.x86_64。 |

# 2 Linux Containers 实现原理

Linux Containers 主要通过 Control Groups（Cgroups）、Namespaces、SELinux 大三技术（由 linux kernel 提供）来实现：

1) Cgroup ：对 container 的资源（CPU、Memory 等）进行控制。
2) Namespace：对 container 中的资源（Net、PID 等）进行隔离。
3) SELinux：对 container 进行安全隔离。

Linux Containers 的架构如下所示：




## 2.1 Cgroup

Linux Containers 依赖 Cgroup 技术对 Container 的资源进行控制，Cgroup 有多个子系统组成，每个子系统实现不同的功能，如下所示：

| ID | 子系统 | 说明 |
|---|---|---|
| 1 | blkio | 设置块设备设定输入/输出限制（如物理设备）。 |
| 2 | cpu | 使用调度程序提供对CPU的cgroup任务访问(控制CPU的利用率)。 |
| 3 | cpuacct | 自动生成cgroup中任务所使用的CPU报告。 |
| 4 | cpuset | 为cgroup中的任务分配独立CPU和内存节点。 |
| 5 | devices | 允许或者拒绝cgroup中的任务访问设备。 |
| 6 | freezer | 挂起或者恢复cgroup中的任务。 |
| 7 | memory | 设置每个cgroup的内存限制以及产生内存资源报告 |
| 8 | net_cls | 标记每个网络包以供cgroup方便使用 |

注：
关于 Cgroup 的详细使用方法请参考：https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/pdf/Resource_Management_Guide/Red_Hat_Enterprise_Linux-6-Resource_Management_Guide-en-US.pdf

## 2.2 Namespaces

Linux Containers 依赖 Namespaces 技术对进程进行隔离，通过如下 6 个 namespace 来实现：

1) Mount namespace
用于隔离文件系统挂载相关信息。在 mount namespace 内的进程进行 mount/unmount 操作将只会在该 mount namespace 内可见，因此可为进程提供独有的文件系统层次结构。

2) Uts namespace
用于隔离 Container OS 版本相关信息。每个 uts namespace 拥有自己的 ostype, osrelease, version, hostname, domainname。

3) Ipc namespace
用于隔离进程间的通信。处于同一 ipc namesapce 的进程才可以互相通信，由于不同 Container 不在同一 ipc namespace，因此不同 Container 中的进程无法互相通信。

4) Net namespace
用于隔离网络相关的资源。每个 net namespace 拥有自己的 net device、IP address、防火墙规则、路由规则等。

5) Pid namespace
用于隔离进程的 PID。Host 和 Container 中可以存在同样的 PID，在 Container 只能查看 Container 中的进程，无法查看其它 Container 或者 Host 上的进程。但在 Host 上可以查看到 Container 下的进程，不过它们被分配不同的 PID。

注：pid namespace 在内核中实现为进程分层结构。如：父 pid namespace 可以看到子 pid namespace 的所有进程，但子 pid namespace 看不到父 pid namespace 的信息。在创建进程时，除了在进程所属的 pid namespace 中申请 pid 外，还需要在父 pid namespace 中申请 pid。

6) User namespace
主要是为了解决安全问题。通过将 Container 上的 root 用户映射为 Host 上的普通用户，可以防止 Container 上的 root 用户进行加载/卸载模块等会对 Host 造成影响的操作。

注：Namespace 相关的 System Call 有 clone、unshare、setns。

## 2.3 SELinux

1) SELinux（Security-Enhance Linux）是 Linux 内核针对 MAC（mandatory access control）机制、MLS（multi-level security）、MCS（multi-category security）实现的一个机制。

2) 仅通过 Cgroup 和 Namespaces 无法保证 Container 中的 root 进程对 Container 外部的进程进行“干涉”，这时候就需要 SELinux 机制来对 Container 进行安全隔离。

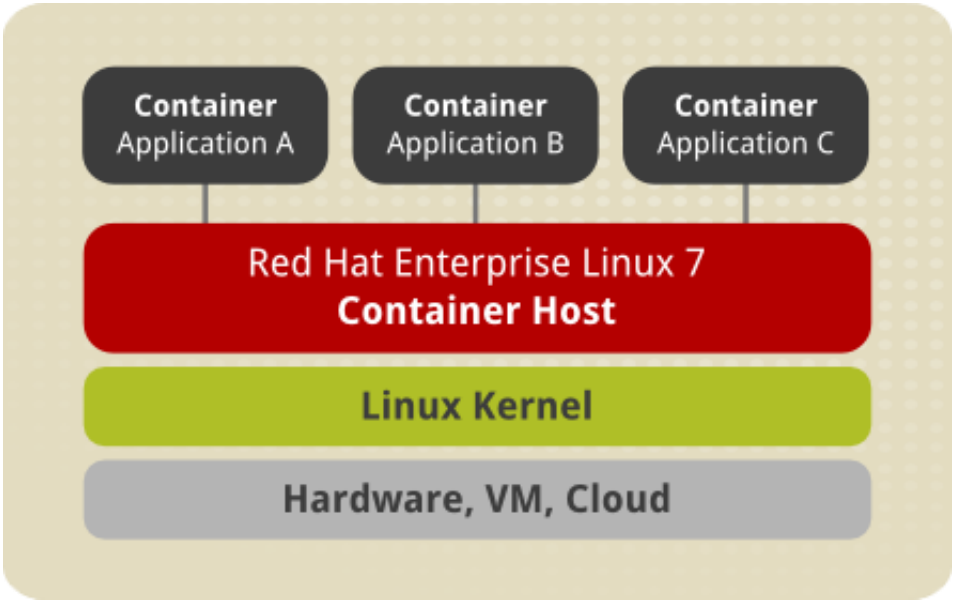
3) Container 在被创建的时候，会根据 SElinux policy 自动为 Container 分配一个 Selinux Context。

注：
关于 SELinux 的详细信息，可参考：
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/SELinux_Users_and_Administrators_Guide/index.html

# 3 Libvirt LXC

## 3.1 介绍

1) Libvirt LXC 相当于一个管理接口可对 Container 进行管理，其架构如下所示：




说明：
a. Container 里运行的 APP 都是基于 OS（如，RHEL7）的用户空间（user space）和运行时（run time）环境。
b. Libvirt LXC 管理的 Container 优点是通过 yum update 命令可以方便的对 Container 进行安全或者其它更新。

2) Libvirt LXC 管理的 Container 有两种类型，分别称为 APP 和 OS，其对比如下：

| 类型 | 独立的根文件系统 | 独立的 CPU | 独立的内存 | 独立的网络 |
|---|---|---|---|---|
| APP | × | ○ | ○ | ○ |
| OS | ○ | ○ | ○ | ○ |

3) Libvirt LXC 启动 Container 的“init”进程不同，又可分为 bash 模式和 init 模式。

4) Libvirt LXC 启动 Container 的方式不同，又可分为临时性（Temporary）Container 和永久性（Persistent）Container。

## 3.2 配置

### 3.2.1 APP Container(bash 模式)

配置 APP Container 可参考如下 xml 文件：

```
<domain type='lxc'>
  <name>app-container</name>
  <uuid>be8f8cc0-1897-5ff9-445b-016323ee81d7</uuid>
  <memory unit='KiB'>524288</memory>
  <currentMemory unit='KiB'>524288</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <os>
    <type arch='x86_64'>exe</type>
    <init>/bin/sh</init>
  </os>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/libvirt_lxc</emulator>
    <interface type='network'>
       <mac address='00:16:3e:bb:7f:b6'/>
       <source network='default'/>
    </interface>
    <interface type='bridge'>
         <mac address='00:17:3e:bb:7f:b6'/>
         <source bridge='br0'/>
    </interface>
    <console type='pty'>
       <target type='lxc' port='0'/>
    </console>
  </devices>
</domain>
```

注：
Container 使用/bin/bash 进程作为 init 进程(PID 为 1)的话，那么退出 shell 时也会关闭 Container。

### 3.2.2 OS Container(bash 模式)

1) 为 Container 准备根文件系统，可参考如下命令创建：

```
#for cur in boot usr opt run bin sbin etc usr lib lib64 var tmp ;do cp -a/$cur
<target_path>/; done
#for cur in home root;do mkdir -p <target_path>/$cur; done
```

注： <target_path>为事先创建的有足够空间的空目录，将作为 Container 根文件系统目录。

2) 配置 APP Container 可参考如下 xml 文件：

```
<domain type='lxc'>
  <name>os-container</name>
  <uuid>1fbbabc1-1775-3429-950f-70c4bee1aa7d</uuid>
  <memory unit='KiB'>524288</memory>
  <currentMemory unit='KiB'>524288</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <os>
    <type arch='x86_64'>exe</type>
    <init>/bin/sh</init>
  </os>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/libexec/libvirt_lxc</emulator>
    <filesystem type='mount' accessmode='passthrough'>
      <source dir='<target_path>'/>
      <target dir='/'/>
    </filesystem>
    <interface type='network'>
      <mac address='52:54:00:23:9a:1f'/>
      <source network='default'/>
    </interface>
    <console type='pty'>
      <target type='lxc' port='0'/>
    </console>
  </devices>
</domain>
```

注：

1) <target_path>为 Container 的根文件系统目录。
2) 使用 service 命令方式会启动不起来相关服务（如：service mysqld start），直接使用二进制文件可启动相关服务（如：/usr/bin/mysqld_safe）。
3) 目前在配置文件设置 Container 的 CPU 个数并不能生效，可在启动 Container 后，再使用 Cgroup（RHEL7 系下是/sys/fs/cgroup/*）对 Containers 的资源（CPU、Memory 等）进行控制。
4) 如果启动 Container 后发现需要给 Container 安装某些包，可直接在 Host 运行 yum 命令进行安装（如：yum –installroot <target_path> install Package_Name）。

### 3.3.3 OS Container(init 模式)

1) 在 Host 上配置 yum 源。

2) 修改脚本 build_custom_rootfs.sh（见附录）中 TARGET_DIR 变量，然后执行脚本创建根文件系统并设置 root 用户密码，命令行如下所示：

```
# ./build_custom_rootfs.sh
# chroot $<TARGET_DIR>
# passwd root
```

3) 配置 APP Container 可参考如下 xml 文件：

```
<domain type='lxc'>
  <name>os-container</name>
  <uuid>1fbbabc1-1775-3429-950f-70c4bee1aa7d</uuid>
  <memory unit='KiB'>524288</memory>
  <currentMemory unit='KiB'>524288</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <os>
    <type arch='x86_64'>exe</type>
    <init>/sbin/init</init>
  </os>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/libexec/libvirt_lxc</emulator>
    <filesystem type='mount' accessmode='passthrough'>
      <source dir='< TARGET_DIR >'/>
      <target dir='/'/>
    </filesystem>
    <interface type='network'>
      <mac address='52:54:00:23:9a:1f'/>
      <source network='default'/>
    </interface>
    <console type='pty'>
      <target type='lxc' port='0'/>
    </console>
  </devices>
</domain>
```

## 3.3 管理

1) Libvirt LXC 创建的 Container 是通过 virsh 命令进行管理的。为了使得 virsh 能够执行管理 Container 相关的命令，libvirt 必须能够连接到 LXC driver（驱动名为：libvirt-daemon-driver-lxc-version）。

2) 在 RHEL7 系下，libvirt 默认是连接到 KVM 的（每个 Host 只能有一个默认的 libvirt URI），为了使得 libvirt 能够连接到 LXC driver，可以使用如下两种方法：

| Method | Description |
|---|---|
| virsh -c lxc:/// command | 临时性的，每次运行 command 时都需要使用参数“virsh -c lxc:/// command”。 |
| export LIBVIRT_DEFAULT_URI=lxc:/// | 永久性的，以后运行 command 可以直接使用”virsh command”即可。 |

注：可以通过命令“virsh uri”来查看当前 libvirt 的 URI。

3) virsh 管理 Container 的常用命令如下所示：

| Command | Description |
|---|---|
| virsh -c lxc:/// define <domain.xml> | 创建Container |
| virsh -c lxc:/// start <domain-name> | 启动Container |
| virsh -c lxc:/// autostart <domain-name> | 开机自动启动Container |
| virsh -c lxc:/// create <domain.xml> | 创建并启动Container（临时性的） |
| virsh -c lxc:/// shutdown <domain-name> | 正常关闭Container |
| virsh -c lxc:/// destroy <domain-name> | 强行关闭Container |
| virsh -c lxc:/// console <domain-name> | 连接Container |
| virsh -c lxc:/// undefined <domain-name> | 删除Container |
| virsh -c lxc:/// edit <domain-name> | 修改Container的xml文件 |
| virsh -c lxc:/// dominfo <domain-name> | 显示Container相关信息 |
| virsh -c lxc:/// list -all | 显示Container |
| virsh -c lxc:/// net-list | 显示virtual network |

注：domain-name 为<domain.xml>文件里面<name>配置的名字。

# 4 附录

脚本 build_custom_rootfs.sh：

```
#!/bin/sh

TARGET_DIR="/mnt"
CUR_DIR=`pwd`

yum --installroot $TARGET_DIR install filesystem setup rpm selinux-policy
systemd dbus initscripts util-linux pam passwd crontabs kmod logrotate rsyslog
openssh openssh-server chkconfig authconfig glibc mailcap net-tools mysql httpd
sysstat java-1.7.0-openjdk libvirt-java libvirt-java-devel javapackages-tools
tzdata-java  mysql-server gcc vim psmisc less tar apr apr-util apr-devel apr-
util-devel pcre-devel bison flex openssh-clients file vim-minimal time mailx bc
tar postfix mariadb*

cd "$TARGET_DIR""/etc/systemd/system/"
ln -s ../../../dev/null sysinit.target
ln -s ../../../dev/null console-shell.service
ln -s ../../../dev/null fedora-readonly.service
ln -s ../../../dev/null fedora-storage-init.service
#ln -s /dev/null "$TARGET_DIR""/etc/systemd/system/"
cd $CUR_DIR

#systemd_tune
cat > "$TARGET_DIR""/etc/systemd/system/basic.target" <<END
[Unit]
Description=Basic System
Documentation=man:systemd.special(7)
Requires=systemd-tmpfiles-setup.service sockets.target
After=systemd-tmpfiles-setup.service sockets.target
RefuseManualStart=yes
END

#ln -s /usr/lib/systemd/system/getty\@.service
"$TARGET_DIR""/etc/systemd/system/getty.target.wants/getty\@tty1.service"
mkdir -p "$TARGET_DIR""/etc/systemd/system/getty.target.wants/"
cd "$TARGET_DIR""/etc/systemd/system/getty.target.wants/"
ln -s ../../../../usr/lib/systemd/system/getty\@.service .
cd $CUR_DIR

#eth0_service
cp /etc/sysconfig/network "$TARGET_DIR""/etc/sysconfig/network"
cat > "$TARGET_DIR""/etc/sysconfig/network-scripts/ifcfg-eth0" <<END
DEVICE=eth0
```

```
BOOTPROTO=dhcp
ONBOOT=yes
NAME=eth0
TYPE=Ethernet
END

cat > "$TARGET_DIR""/etc/systemd/system/lxc-eth0.service" <<END
[Unit]
Before=multi-user.target
Conflicts=shutdown.target
Description=bring up eth0 in this container[Service]
ExecStart=/usr/sbin/ifup eth0
Type=simple
END

#ln -s "$TARGET_DIR""/etc/systemd/system/lxc-eth0.service" ../
#with problem
#ln -s "/etc/systemd/system/lxc-eth0.service"
"$TARGET_DIR""/etc/systemd/system/basic.target.wants/lxc-eth0.service"
mkdir "$TARGET_DIR""/etc/systemd/system/basic.target.wants/" -p
cd "$TARGET_DIR""/etc/systemd/system/basic.target.wants/"
ln -s ../lxc-eth0.service .
cd $CUR_DIR

#empty_fstab
truncate "$TARGET_DIR""/etc/fstab" --size 0

#pam_tune
# to be done
FILE_LIST=`find "$TARGET_DIR""/etc/pam.d"`

for cur in $FILE_LIST;
do
    test -d $cur && continue;
    test -L $cur && continue;
    sed -i "s/^.*pam_loginuid.so.*$/#&/" $cur
done

#securetty_tune
cat >> "$TARGET_DIR""/etc/securetty" <<END
pts/0
END
```