# 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](javascript:void(0);) | Low | High |
| 5 | Complexity | Low | High |

注：本文在如下平台下进行Linux Containers研究及测试。

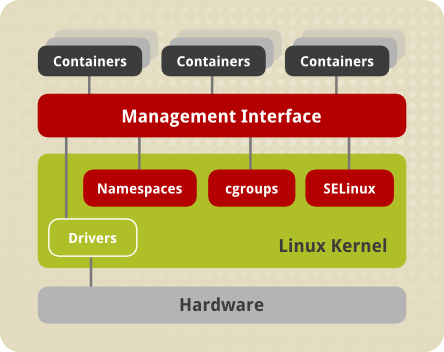
|  |  |
| --- | --- |
| - | 描述 |
| os | RHEL7.0\_x86\_64 |
| kernel | kernel-3.10.0-110.el7.x86\_64 |
| cpu | Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz |
| glibc | glibc-2.17-52.el7.x86\_64。 |

# 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。

1. 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。

1. 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。

1. 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。

1. 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。

1. 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：

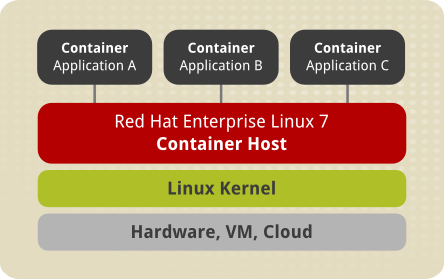
关于SELinux的详细信息，可参考：

<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：

1. 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。
2. 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。
3. 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 | ○ | ○ | ○ | ○ |

1. Libvirt LXC启动Container的“init”进程不同，又可分为bash模式和init模式。
2. 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。

1. 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 |

1. 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。

1. 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 |