## LXC(Linux containers)快速入门

1.LXC是什么？

LXC是Linux containers的简称，是一种基于容器的操作系统层级的虚拟化技术。

2.LXC可以做什么？

LXC可以在操作系统层次上为进程提供的虚拟的执行环境，一个虚拟的执行环境就是一个容器。可以为容器绑定特定的cpu和memory节点，分配特定比例的cpu时间、IO时间，限制可以使用的内存大小（包括内存和是swap空间），提供device访问控制，提供独立的namespace（网络、pid、ipc、mnt、uts）。

3.LXC如何实现？

Sourceforge上有LXC这个开源项目，但是LXC项目本身只是一个为用户提供一个用户空间的工具集，用来使用和管理LXC容器。LXC真正的实现则是靠Linux内核的相关特性，LXC项目只是对此做了整合。基于容器的虚拟化技术起源于所谓的资源容器和安全容器。

LXC在资源管理方面依赖与Linux内核的cgroups子系统，cgroups子系统是Linux内核提供的一个基于进程组的资源管理的框架，可以为特定的进程组限定可以使用的资源。LXC在隔离控制方面依赖于Linux内核的namespace特性，具体而言就是在clone时加入相应的flag（NEWNS NEWPID等等）。

4.为什么要选择LXC？

LXC是所谓的操作系统层次的虚拟化技术，与传统的HAL（硬件抽象层）层次的虚拟化技术相比有以下优势：

1. 更小的虚拟化开销（LXC的诸多特性基本由内核特供，而内核实现这些特性只有极少的花费，具体分析有时间再说）
2. 快速部署。利用LXC来隔离特定应用，只需要安装LXC，即可使用LXC相关命令来创建并启动容器来为应用提供虚拟执行环境。传统的虚拟化技术则需要先创建虚拟机，然后安装系统，再部署应用。

LXC跟其他操作系统层次的虚拟化技术相比，最大的优势在于LXC被整合进内核，不用单独为内核打补丁。

5.如何使用LXC？

**Cgroup和LXC(Linux containers)安装详解(Centos 6.2)**

**系统**：Centos 6.2

**系统依赖**：

gcc：yum install gcc

libcap-devel:yum install libcap-devel

**安装Cgroup**

yum install libcgroup

**安装LXC**

* 从sourceforge上下载源代码:sourceforge.net/projects/lxc/files/lxc
* 解压
* 进入解压后的目录执行以下命令
* ./configure
* make
* make install

**注意:**

make install 后，默认是把lxc相关的命令拷贝到了/usr/local/bin,如果你没有把/usr/local/bin加入到PATH，你使用lxc命令时，系统会说找不到命令。解决办法：在主目录下的.bash\_profile中将/usr/local/bin加入到PATH中。

还有一个问题是第一次执行时，它会说配置目录找不到/usr/local/var/lib/lxc，我看了下系统本身是没有这个目录的，解决办法，建立这个目录：mkdir -p /usr/local/var/lib/lxc

## LXC(Linux containers)常用命令介绍

**lxc-version** 用于显示系统LXC的版本号（可以通过此命令判断系统是否安装了lxc）

用法：lxc-version

例如:lxc-version

**lxc-checkconfig** 用于判断linux内核是否支持LXC

用法：lxc-checkconfig

例如：lxc-checkconfig

**lxc-create**用于创建一个容器

用法：lxc-create -n name [-f config\_file]

         -n 后面跟要创建的容器名字 例如：-n foo

         -f 后面跟容器配置文件的路径

注：1.采用lxc-create创建的容器，在停止运行后，不会被销毁，要采用lxc-destroy命令才能销毁

       2.容器命令空间是全局的，系统中不允许存在重名的容器，如果-n 后面跟一个已经存在的容器名，创建会失败

例如：lxc-create --n foo --f foo.conf

**lxc-execute** 用于在一个容器执行应用程序

用法： lxc-execute -n name [-f config\_file] [ -s KEY=VAL ]command

           -n 后面跟容器名字（容器名字用于管理容器）例如：-n foo

           -f 后面跟容器配置文件的路径（如果没有配置文件，可以直接用-s指定配置选项，如果什么都没有，系统采用默认策略）例如：-f foo.conf

           -s 后面跟配置键值对 例如：lxc.cgroup.cpu.shares=512

       command 为要执行的命令 例如：/bin/bash

 注：1.如果容器还不存在，lxc-execute会自动创建一个,容器停止运行后会被自动销毁

        2.用lxc-execute启动应用程序，配置优先级如下：

          如果指定-f选项，那么之前创建容器（如果容器是已存在的）的配置文件不会被使用

          如果没有指定-f选项，那么将采用之前创建容器（如果容器已存在）的配置文件

          如果指定-s选项，则在命令行中的配置键值对会覆盖配置文件（无论之前的还是-f指定的）相同配置

例如：lxc-execute --n foo --s lxc.cgroup.cpu.shares=512 /bin/bash

**lxc-start** 用于在容器中执行给定命令

用法：lxc-start  -n name [-f config\_file] [-c console\_file] [-d] [-s KEY=VAL]  
          [command]

      -d 将容器当做守护进程执行

     -f  后面跟配置文件

     -c 指定一个文件作为容器console的输出，如果不指定，将输出到终端

    -s 指定配置

例如：lxc-start -n foo -f foo.conf -d /bin/bash

注：1.如果容器还不存在，lxc-start会自动创建一个,容器停止运行后会被自动销毁

       2.lxc-start配置优先级与lxc-execute相同

       3.lxc-start 与lxc.execute的异同：

        lxc-start 和 lxc-execute都可以在容器中启动进程，区别在于lxc-start直接创建进程，lxc-execute先创建lxc-init进程，然后在lxc-init中fork一个进程来执行。

       The orphan process group and daemon are not supported by this  command,  
       use the lxc-execute command instead.

       If no command is specified, lxc-start will use the default "/sbin/init"  
        command to run a system container.

      4.lxc-start用于在容器启动system，lxc-execute用于在容器执行应用程序

**lxc-kill** 发送信号给容器中的第一个用户进程（容器内部进程号为2的进程）

用法：lxc-kil -n name SIGNUM

      -n 后面跟容器名

      SIGNUM  信号 （此参数可选，默认SIGKILL）

例如：lxc-kill -n foo

**lxc-stop** 用于停止容器中所有的进程

用法：lxc-stop -n name

       -n后面跟要停止的容器名

例如:lxc-stop --n foo

**lxc-destroy** 用于销毁容器

用法：lxc-destroy -n name

       -n后面跟要停止的容器名

例如: lxc-destroy --n foo

**lxc-cgroup** 用于获取或调整与cgroup相关的参数

用法：lxc-cgroup -n name subsystem value

        -n 后面跟要调整的容器名

例如： lxc-cgroup -n foo devices.list

           lxc-cgroup -n foo cpuset.cpus "0,3"

**lxc-info** 用户获取一个容器的状态

 用法:lxc-info -n name

       -n后面跟操作的容器名

例如: lxc-info --n foo

注：容器的状态有：STARTING RUNNING STOPPING STOPPED ABORTING

**lxc-monitor** 监控一个容器状态的变换，当一个容器的状态变化时，此命令会在屏幕上打印出容器的状态

用法:lxc-monitor -n name

例如：lxc-monitor -n foo

**lxc-ls** 列出当前系统所有的容器

用法：lxc-ls

例如：lxc-ls

**lxc-ps** 列出特定容器中运行的进程

用法:lxc-ps

例如:lxc-ps -n foo

更多的命令介绍可以再man page获得

## LXC(Linux containers)配置文件简介

LXC配置文件的man page ：<http://manpages.ubuntu.com/manpages/natty/man5/lxc.conf.5.html>

LXC配置项都是以key=value的形式，#开始的一行代码注释。配置也可以在lxc-execute或者lxc-start的命令行以-s key=value设定

这里介绍一下常用的配置：

**1.Cgroup部分（所有Cgroup子系统的配置参数均可以通过lxc-cgroup 命令进行动态获得或者调整）**

LXC采用Cgroup来对容器进行资源管理，因此Cgroup的各个子系统均可以使用。实际上使用LXC之前，必须执行mount none -t cgroup /cgroup 挂载cgroup，或者采用echo "none /cgroup cgroup defaults 0 0" >> /etc/fstab让系统每次启动自动挂载Cgroup。这样以来Cgroup的所有子系统自动挂载到了一个hierarchy，LXC在创建容器的时候就在/ group下创建一个子group以实现对容器的资源控制，我们就可以根据需要设定相应子系统的参数来达到目的。

Cgroup部分的配置都是以lxc.cgroup.[subsystem name].key=value的形式出现的.例如：lxc.cgroup.cpu.shares=512 lxc.cgroup.cpuset.cpus=1.2

下面介绍下用到的Cgroup的子系统的用法（更多关于Cgroup的介绍，请到RedHat官方文档网上下载）

**cpu 子系统**调度对 cgroup 的 CPU 访问。可根据以下参数调度对 CPU 资源的访问，每个参数都独立存  
在于 cgroup 虚拟文件系统的伪文件中：

cpu.shares  
包含用来指定在 cgroup 中的任务可用的相对共享 CPU 时间的整数值。例如：在两个 cgroup 中都  
将 cpu.shares 设定为 1 的任务将有相同的 CPU 时间，但在 cgroup 中将 cpu.shares 设定为 2  
的任务.

特别说明：根据实验结果以及源代码分析，根目录的shares值默认是1024，同一层次的task和task group平等的接受调度。也就说在根group下有task A 子group B 子group C，如果B的shares值为512，C的shares值为1024，那么task A 和group B 的进程和 group C中的进程CPU份额大致应该是2:1:2。（当然如果task或者group B用不到那么多份额，会自动让给其他task或group）

**cpuset 子系统**为 cgroup 分配独立 CPU 和内存节点。可根据用以下参数指定每个 cpuset，每个参数  
都在控制组群虚拟文件系统中有单独的伪文件：

cpuset.cpus（强制）  
指定允许这个 cgroup 中任务访问的 CPU。这是一个用逗号分开的列表，格式为 ASCII，使用小横  
线（"-"）代表范围，例如：cpuset.cpus=1,2-4

**特别说明：单线程程序即使分配多个核心也只会用一个，对于有fork调用的程序，系统会做SMP均衡，但是所有进程仍然在限定的CPU上运行。**

cpuset.mems（强制）  
指定允许这个 cgroup 中任务可访问的内存节点。这是一个用逗号分开的列表，格式为 ASCII，使  
用小横线（"-"）代表范围,例如：cpuset.mems=1,3-5

特别说明：此选项在SMP架构的机器不需要设置，机器只有一个内存节点。主要针对NUMA的机器使用

**memory** 子系统自动生成 cgroup 中任务使用的内存资源报告，并设定由那些任务使用的内存限制：

memory.limit\_in\_bytes  
设定用户内存的最大量（包括文件缓存）。如果没有指定单位，则将该数值理解为字节。

特别说明：在有swap空间的机器上单独设置这个选项，malloc超过限制值，程序仍然能够可以运行，原因在于程序将内存的数据交换到了swap空间，不会超限。可以使用/sbin/swapoff关掉swap空间（不推荐）

memory.memsw.limit\_in\_bytes  
设定最大内存与 swap 用量之和。如果没有指定单位，则将该值解读为字节。

特别说明：设置这个选项之前，必须设定memory.limit\_in\_bytes，且memory.limit\_in\_bytes必须小于或等于此选项的值。在有swap空间的机器上，设置了这个选项，可以严格控制程序对内存的使用，只要malloc超过这个值，就会被系统kill掉

**blkio子系统**

块 I/O（blkio）子系统控制并监控 cgroup 中的任务对块设备的 I/O 访问。在部分这样的伪文件中写入值可限制访问或者带宽，且从这些伪文件中读取值可提供 I/O 操作信息

blkio.weight  
指定 cgroup 默认可用访问块 I/O 的相对比例（加权），范围在 100 到 1000。这个值可由具体设备的 blkio.weight\_device 参数覆盖。此数值跟cpu.share类似，以访问时间的比例的形式起作用。默认根group的值为1000.如果再根下面建立两group，分别设置weight为500和1000，那么IO时间比值为1:2.更多的说明可以参考内核文档。

 blkio.weight\_device  
指定对 cgroup 中可用的具体设备 I/O 访问的相对比例（加权），范围是 100 到 1000。这个值的格式为major:minor weight，其中 major 和  
minor 是在《Linux 分配的设备》中指定的设备类型和节点数，我们也称之为 Linux 设备列表，您  
可以参考 <http://www.kernel.org/doc/Documentation/devices.txt>

blkio.throttle.read\_bps\_device

针对特定的 major:minor 设备的读bps的上限限制（bps指byte per second，即每秒的byte数)

blkio.throttle.write\_bps\_device

针对特定的 major:minor 设备的写bps的上限限制

 blkio.throttle.read\_iops\_device

针对特定的 major:minor 设备的读ios的上限限制(ios指io per second,即每秒的io次数)

blkio.throttle.write\_iops\_device

针对特定的 major:minor 设备的写ios的上限限制

此四个参数可以同blkio.weight blkio.weight\_device以前使用，系统将同时满足着些限制。

**2.network部分**

network部分的配置均以lxc.network开头.

**lxc.network.type指定用于容器的网络类型**：

empty：新的网络空间会为容器建立，但是没有任何网络接口

veth：新的网络空间会为容器建立，容器将链接到lxc.network.link定义的网桥上，从而与外部通信。网桥必须在预先建立。

macvlan：新的网络空间会为容器建立，一个macvlan的接口会链接到lxc.network.link上。

phys：新的网络空间会被建立，然后lxc.network.link指定的物理接口会被分配给容器

**lxc.network.flags用于指定网络的状态**

up：网络处于可用状态

**lxc.network.link**用于指定用于和容器接口通信的真实接口，比如一个网桥br0

**lxc.network.name**用于指定容器的网络接口的名字，与宿主接口不在一个命名空间。如果不指定，lxc为自动分配一个

**lxc.network.hwaddr**用于指定容器的网络接口的mac地址

**lxc.network.ipv4**用于指定容器的网络接口的ipv4地址，如果不设置或者设为0.0.0.0则表示，容器将通过dhcp的方式获得ip地址

**lxc.network.ipv6**用于指定容器的网络接口的ipv6地址

**3.rootfs部分**

lxc.rootfs用于指定容器的虚拟根目录，设定此项以后，容器内所有进程将会把此目录根目录，不能访问此目录之外的路径，相当于chroot的效果

PS：用这几个部分已经可以配一个比较完整的应用容器了，提供了cpu memory IO资源控制，提供单独的网络、根目录空间，再结合Loop文件或者LVM做一下硬盘限制就更好了。

## [LXC 网络配置实例（Redhat）](http://www.cnblogs.com/lisperl/archive/2012/04/19/2456797.html)

LXC为容器提供了独立的网络空间，这意味着我们可以为容器配置IP地址，MAC地址，route表等等，并通过LXC提供的几种方式连接到宿主进而连到互联网。下面介绍两种采用网桥方式连接到宿主机的配置情况。

情况一：

采用静态IP地址，即宿主本身的IP地址是由系统管理员手动配置的，容器的IP地址也是静态指定。

宿主机的配置：

vim /etc/sysconfig/network-scripts/ifcfg-eth0

DEVICE=eth0

ONBOOT=yes

BOOTPROTO=none

USERCTL=no

BRIDGE=br0

vim /etc/sysconfig/network-scripts/ifcfg-br0

DEVICE=br0

TYPE=Bridge

BOOTPROTO=none

ONBOOT=yes

DELAY=0

IPADDR=10.20.153.21

NETMASK=255.255.255.0

GATWAY=10.20.153.254

然后重启网络

service network restart

下面来配置LXC容器

vim net.conf

lxc.network.type=veth

lxc.network.flags=up

lxc.network.link=br0

lxc.network.ipv4=10.20.153.234/24

lxc.network.hwaddr=4a:49:43:49:79:bd

然后创建LXC容器

lxc-create -n net -f net.conf

最后就可使用LXC容器来启动应用了

lxc-execute -n net ping 10.20.153.20

情况二：

采用Dhcp+NAT的方式，即宿主机的IP地址由dhcp协议自动获取，LXC容器的IP地址也通过dhcp协议获取

宿主机配置：

yum install libvirt

这时机器上会多出一个virbr0的网桥，通过ifconfig就可以看到

LXC容器的配置：

lxc.network.type=veth

lxc.network.flags=up

lxc.network.link=virbr0

lxc.network.hwaddr=4a:49:43:49:79:bd

然后创建LXC容器

lxc-create -n net -f net.conf

最后使用容器启动应用，注意此时不能直接启动应用，需要先一步使用dhcp获得IP地址，可以写一个脚本来替代。比如：

vim net.sh

dhclient eth0

ping xx.xx.xx.xx

再启动容器：

chmod +x net.sh

lxc-execute -n net ./net.sh

注:如果是ubuntu系统，通过apt-get 安装的lxc，不用再安装libvirt，安装好lxc后，系统会多一个lxcbr0,功能与virbr0类似