.Deb

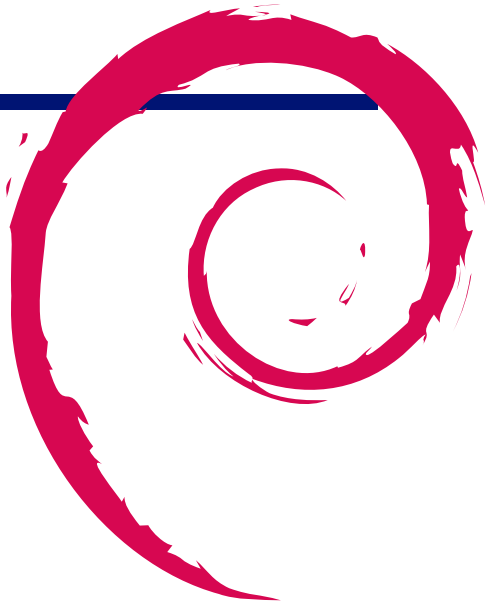
銀河系唯一のDebian専門誌

2016 年 8 月 20 日

特集：LXC

デビアン勉強会

目次



1 最近の Debian 関連のミーティング報告

杉本 典充

1.1 第 141 回東京エリア Debian 勉強会

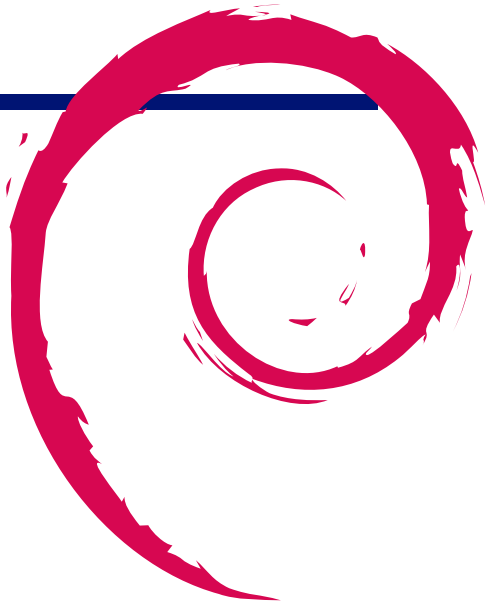
2016 年 7 月 16 日 (土) に第 141 回東京エリア Debian 勉強会を開催しました。会場は日本橋にあるサイボウズさんをお借りして行いました。参加者は 12 名でした。発表は、Roger さんによる「Debian Installer Screen 対応」と yutannihilation さんによる「go-apt-cacher/mirror」の 2 つでした。

Roger さんによる「Debian Installer Screen 対応」の発表は、Debconf16 で発表された内容と同一であり、ディスプレイを持たない（いわゆるヘッドレス）マシンにおいて、Debian Installer で表示しているコンソール、シェル、ログを screen で表示の切り替えをできるようにする開発作業の報告でした。Debian Installer で利用される udeb パッケージの説明と udeb パッケージに新たなプログラムを追加する過程の説明がありました。成果については、Debian Installer Stretch Alpha 7 に取り込まれて配布が開始されています。

yutannihilation さんによる「go-apt-cacher/mirror」の発表は、apt リポジトリをキャッシュするツールとミラーするツールを Go 言語で作ってみたという発表でした[*1]。apt リポジトリをミラーするプログラム「apt-mirror」はすでにありますが、ファイルのハッシュ確認をしていないため壊れたミラーが生成されてしまうことがあったとのことです。これらを解決すべく、新規に apt リポジトリをミラーおよびキャッシュするプログラムを開発したとのことでした[*2]。go-apt-cacher/mirror は従来ツールに比べ高速に動作するとのことで、非常に有用なツールになりそうです。

[*1] `https://github.com/cybozu-go/aptutil`

[*2] 既存プログラムは歴史があるプログラムのため、構造上改修するのがなかなか難しいとのことで、新規開発することにした経緯があるとのことです。



2 事前課題

杉本 典充

今回の事前課題は以下です:

1. コンテナ型仮想化技術 (LXC/LXD/Docker/etc) 使ったことありますか？
2. 先の質問で「あります」とお答えした方にお聞きします。そのコンテナ型仮想化技術をお教えください。
3. Hack Time は何をしますか。

この課題に対して提出いただいた内容は以下です。

2.1 mkouhei

1. ある
2. LXC
3. パッケージメンテナンス

2.2 dictoss

1. ある
2. lxc
3. preseed、kfreebsd の動向調査

2.3 かい

1. なし
2. (回答なし)
3. .deb をまだ作ったことがないので作ってみる

2.4 kenhys

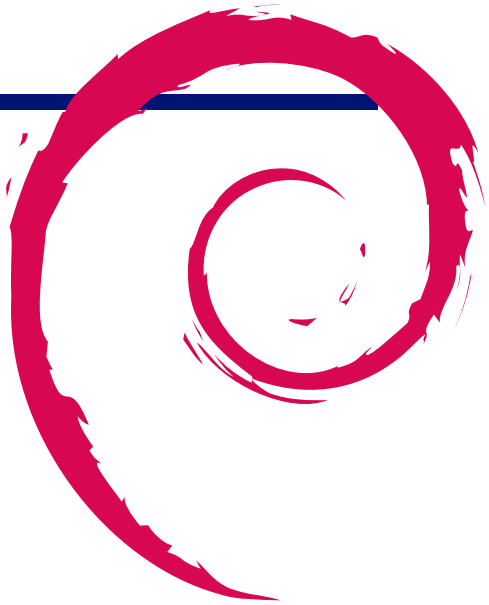
1. ある
2. Docker
3. 未定

2.5 koedoyoshida

1. ある
2. LXC,Docker お試し程度、chroot の方が使ってる
3. 未定

2.6 yy_y_ja_jp

1. ない
2. cowbuilder?
3. DDTSS?



3 Debian Trivia Quiz

杉本 典充

Debian の昨今の話題についての Quiz です。

今回の出題範囲は debian-devel-announce@lists.debian.org や debian-news@lists.debian.org に投稿された内容などからです。

問題 1. unstable にて GCC が Transition されました。さて、バージョンは何に Transition されたでしょうか

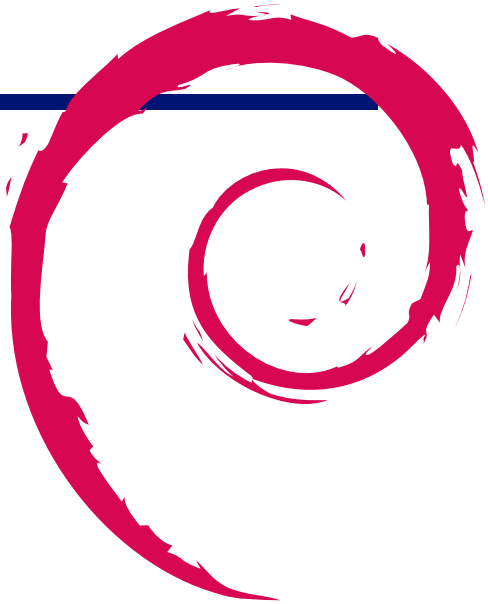
☐ A 5.4
☐ B 6.1
☐ C 7.0

問題 2. unstalbe にて perl が Transition されました。さて、バージョンは何に Transition されたでしょうか

☐ A 5.22
☐ B 5.24
☐ C 6.0

問題 3. Linux Kernel において次の LTS(=LongTermSupport) とするバージョンの候補が上がってきました。さてバージョンは何でしょうか

☐ A 4.8
☐ B 4.9
☐ C 4.10



4 Debian で lxc をセットアップしてみよう

杉本 典充

4.1 はじめに

コンピュータの仮想化技術にコンテナ型仮想化という仕組みがあります。本発表では、コンテナ型仮想化技術の 1 つである lxc について説明します。

4.2 仮想化技術について

4.2.1 仮想化技術の分類

コンピュータにおける仮想化技術にはいくつかの種類があります。

- コンテナ型仮想化
  - ゲスト環境はホスト環境のあるディレクトリに配置した実行ファイルおよびライブラリの集合であり、実行中のゲスト環境はホスト環境から見ると単なるプロセス群である。実装例は OpenVZ、LXC。
- 準仮想化型
  - ホスト環境とやりとりする API を利用できるように修正が入った OS をゲスト環境として動作させる方式 (=既存の OS そのままでは動かない)。実装例は Xen。
- 完全仮想化型 (エミュレーション型)
  - 既存の OS を無修正のままゲスト環境として動作させる。ホスト環境上で動作する仮想化アプリケーションがゲスト環境をエミューションする。実装例は QEMU、VirtualBox。
- 完全仮想化型 (ハイパーバイザ型)
  - 既存の OS を無修正のままゲスト環境として動作させる。CPU の仮想化機能を使うことでホスト環境におけるオーバーヘッドを極力減らし、エミュレーション型より高いパフォーマンスを出せる。実装例は KVM。

4.2.2 仮想化技術のメリットとデメリット

準仮想化型および完全仮想化型におけるゲスト環境は、特定のハードウェアをもつ物理マシンをエミュレートするいわゆる「仮想マシン」を実行の単位とするため、仮想サーバ上でもカーネルを実行させる必要があることから CPU、メモリ、ディスクを多く使用します。その代わり、物理マシンへ普通に OS をインストールして実行していたプログラムをそのまま実行できる場合が非常に多いことが優位です。

コンテナ型仮想化におけるゲスト環境は、ホスト環境のカーネルで動作するプロセス群であるため、余分なカーネルや管理デーモンを動作させる必要がなく省リソースで動作できる点が優位です。その代わり、ホスト環境の OS や

設定によってゲスト環境に動作制約がつくことがあります[*3]。

4.2.3 コンテナ型仮想化技術の基本 chroot

コンテナ型仮想化の基礎技術に chroot があります[*4]。あるディレクトリ配下に実行ファイル、ライブラリ、設定ファイルを適切に配置した rootfs(=コンテナ環境) に対して chroot(2) または chroot(3) を実行することで、コンテナ環境内のライブラリを使ってコマンドを実行できます。

4.3 lxc 解説

4.3.1 lxc とは

lxc とは[*5]、コンテナ型仮想化として動作する単位であるゲスト環境 (=rootfs) の配置、起動、終了、コンソール入出力、ネットワーク等を管理する仕組みです。lxc を使うことによって、コンテナ環境を起動し、IP アドレスを割り当て、あたかもそこに 1 台の仮想マシンが起動したかのように振る舞うことができます。

4.3.2 lxc のインストール

今回は Debian GNU/Linux 8 Jessie の amd64 上で lxc をインストールして動作させてみます。Debian Project では以下にドキュメントがまとまっています。

- https://wiki.debian.org/LXC
- https://wiki.debian.org/LXC/SimpleBridge

まず、lxc パッケージと仮想ネットワークを構築するパッケージをインストールします

```
# apt-get install lxc bridge-utils libvirt-bin
```

lxc をインストールすると、lxc-*コマンドが使えるようになります。

```
$ ls /usr/bin/lxc*
/usr/bin/lxc-attach       /usr/bin/lxc-start                   /usr/bin/lxc-test-list
/usr/bin/lxc-autostart    /usr/bin/lxc-start-ephemeral         /usr/bin/lxc-test-locktests
/usr/bin/lxc-cgroup       /usr/bin/lxc-stop                    /usr/bin/lxc-test-lxcpath
/usr/bin/lxc-checkconfig  /usr/bin/lxc-test-apparmor           /usr/bin/lxc-test-may-control
/usr/bin/lxc-clone        /usr/bin/lxc-test-attach             /usr/bin/lxc-test-reboot
/usr/bin/lxc-config       /usr/bin/lxc-test-autostart          /usr/bin/lxc-test-saveconfig
/usr/bin/lxc-console      /usr/bin/lxc-test-cgpath             /usr/bin/lxc-test-shutdowntest
/usr/bin/lxc-create       /usr/bin/lxc-test-clonetest          /usr/bin/lxc-test-snapshot
/usr/bin/lxc-destroy      /usr/bin/lxc-test-concurrent         /usr/bin/lxc-test-startone
/usr/bin/lxc-device       /usr/bin/lxc-test-console            /usr/bin/lxc-test-symlink
/usr/bin/lxc-execute      /usr/bin/lxc-test-containertests     /usr/bin/lxc-unfreeze
/usr/bin/lxc-freeze       /usr/bin/lxc-test-createtest         /usr/bin/lxc-unshare
/usr/bin/lxc-info         /usr/bin/lxc-test-destroytest        /usr/bin/lxc-usernsexec
/usr/bin/lxc-ls           /usr/bin/lxc-test-device-add-remove  /usr/bin/lxc-wait
/usr/bin/lxc-monitor      /usr/bin/lxc-test-get_item
/usr/bin/lxc-snapshot     /usr/bin/lxc-test-getkeys
```

libvirt-bin をインストールして libvirtd を起動すると、デフォルトで 192.168.122.0/24 の NAT ネットワークが構成されます。今回は作成したコンテナ環境をこの NAT ネットワークに接続し、ホスト環境とつながる 192.168.122.1(=virbr0) をデフォルトゲートウェイとして通信できるようにします。[*6]

次に、lxc で動作するコンテナ環境にリソース制約を行う仕組みである cgroups を設定します。

[*3] raw ソケットが利用できない、共有メモリが利用できない、loopback が利用できない、同一ホスト環境で動作している他のコンテナと通信できてしまう、などの制約がある場合があります。

[*4] chroot システムコールは 1982 年にウィリアム・ネルソン・ジョイ (ビル・ジョイの名で知られている) が開発したことを起源とされています。ビル・ジョイはプログラムをクリーンビルドできる環境がほしかったため通常利用の環境と分離する手段として開発したと言われています。

[*5] LinuX Containers を省略して lxc と読んでいます

[*6] 仮想ネットワークはノートパソコンや VPS で構築する場合は NAT の方がよいかもしれませんが、社内ネットワークで利用する場合はブリッジ接続 (=ホスト環境とコンテナ環境が同一ネットワークに属する形態) にすることを考えた方がよいかもしれません。

```
# vi /etc/fstab
cgroup  /sys/fs/cgroup  cgroup  defaults  0   0

# mount /sys/fs/cgroup
# mount | grep cgroups
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,
release_agent=/lib/systemd/systemd-cgroups-agent,name=systemd)
```

この状態で、lxc を使える環境であるか確認するコマンド”lxc-checkconfig”を実行してみます。「enabled」と書かれているところは、実際のターミナル上では緑色で表示されます。disable の機能はありませんので、動作できる環境であることが確認できました。

```
# lxc-checkconfig
Kernel configuration not found at /proc/config.gz; searching...
Kernel configuration found at /boot/config-3.16.0-4-amd64
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled
Multiple /dev/pts instances: enabled

--- Control groups ---
Cgroup: enabled
Cgroup clone_children flag: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled
Cgroup cpuset: enabled

--- Misc ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
File capabilities: enabled

Note : Before booting a new kernel, you can check its configuration
usage : CONFIG=/path/to/config /usr/bin/lxc-checkconfig
```

4.3.3 **コンテナの作成**:lxc-create

ゲスト環境となるコンテナ環境 (=rootfs) は lxc-create コマンドで作成することができます。今回の lxc のコンテナ環境は、ホスト環境と同じ Debian GNU/Linux 8 Jessie amd64 を作成します[*7]。今回は、”debstudy1” という名前でコンテナを作成します。

```
# LANG=C SUITE=jessie MIRROR=http://ftp.jp.debian.org/debian lxc-create -n debstudy1 -t debian

debootstrap is /usr/sbin/debootstrap
Checking cache download in /var/cache/lxc/debian/rootfs-jessie-amd64 ...
Copying rootfs to /var/lib/lxc/debstudy1/rootfs...Generating locales (this might take a while)...
Generation complete.
insserv: warning: current start runlevel(s) (empty) of script ‘checkroot.sh’ overrides LSB defaults (S).
insserv: warning: current stop runlevel(s) (S) of script ‘checkroot.sh’ overrides LSB defaults (empty).
insserv: warning: current start runlevel(s) (empty) of script ‘checkroot.sh’ overrides LSB defaults (S).
update-rc.d: error: umountfs Default-Start contains no runlevels, aborting.
insserv: warning: current start runlevel(s) (empty) of script ‘hwclock.sh’ overrides LSB defaults (S).
insserv: warning: current stop runlevel(s) (0 6 S) of script ‘hwclock.sh’ overrides LSB defaults (0 6).
update-rc.d: error: cannot find a LSB script for hwclockfirst.sh
Creating SSH2 RSA key; this may take some time ...
2048 df:99:56:34:c7:6d:d1:0a:2d:e2:b4:6a:fd:a0:62:f5 /etc/ssh/ssh_host_rsa_key.pub (RSA)
Creating SSH2 DSA key; this may take some time ...
1024 9d:42:45:1d:fd:03:92:04:6c:e0:fb:e6:06:cc:07:06 /etc/ssh/ssh_host_dsa_key.pub (DSA)
Creating SSH2 ECDSA key; this may take some time ...
256 6a:4a:1a:6f:27:59:33:6c:58:5c:58:27:03:08:3b:ea /etc/ssh/ssh_host_ecdsa_key.pub (ECDSA)
Creating SSH2 ED25519 key; this may take some time ...
256 36:d6:9b:d3:9d:96:a4:af:af:8c:75:11:90:76:56:75 /etc/ssh/ssh_host_ed25519_key.pub (ED25519)
Failed to read /proc/cmdline. Ignoring: No such file or directory
invoke-rc.d: policy-rc.d denied execution of start.

Current default time zone: ’Asia/Tokyo’
Local time is now:      Sun Jul 10 13:26:07 JST 2016.
Universal Time is now:  Sun Jul 10 04:26:07 UTC 2016.

Root password is ’Won4EiUa’, please change !
```

lxc-create を実行すると、”/var/lib/lxc/debstudy1” というディレクトリが作成され、その配下に rootfs と lxc のゲスト環境用設定ファイルが生成されます。なお、一度コンテナを作成するとダウンロードした rootfs ファイルは

[*7] ゲスト環境は、ホスト環境の linux カーネルで動作するバイナリであればホスト環境と異なるディストリビューションでも動作可能です

キャッシュされます。(2 回目以降の作成で Debian パッケージのダウンロードが不要になり早く処理が終わります)

```
# ls -l /var/lib/lxc/debstudy1
合計 8
-rw-r--r--  1 root root  479  7 月 10 13:26 config
-rw-r--r--  1 root root    0  7 月 10 13:26 fstab
drwxr-xr-x 22 root root 4096  7 月 10 13:26 rootfs

# ls /var/lib/lxc/debstudy1/rootfs
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  selinux  srv  sys  tmp  usr  var
```

lxc が参照する設定ファイルの初期状態は以下になります。

```
# cat /var/lib/lxc/debstudy1/config
# Template used to create this container: /usr/share/lxc/templates/lxc-debian
# Parameters passed to the template:
# For additional config options, please look at lxc.container.conf(5)
lxc.network.type = empty
lxc.rootfs = /var/lib/lxc/debstudy1/rootfs

# Common configuration
lxc.include = /usr/share/lxc/config/debian.common.conf

# Container specific configuration
lxc.mount = /var/lib/lxc/debstudy1/fstab
lxc.utsname = debstudy1
lxc.arch = amd64
lxc.autodev = 1
lxc.kmsg = 0
```

この config ファイルにコンテナ環境が利用するネットワーク設定を行います。

```
# vi /var/lib/lxc/debstudy1/config
（末尾に以下を追加）

lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = virbr0
lxc.network.name = eth0
lxc.network.ipv4 = 192.168.122.203/24
lxc.network.ipv4.gateway = 192.168.122.1
```

4.3.4 コンテナの一覧:lxc-ls

lxc-ls コマンドで作成したコンテナの一覧を表示できます。

```
# lxc-ls
debstudy1
```

4.3.5 コンテナの削除:lxc-destroy

lxc-destroy コマンドでコンテナを削除できます。

```
# lxc-destroy -n <lxc-name>
```

4.3.6 コンテナの起動:lxc-start

lxc-start コマンドでコンテナを起動します。コンテナを起動すると、コンテナ環境内にある init プログラムが実行され常駐します。

-d オプションはコンテナ環境をバックグラウンドで起動するオプションです。-d オプションを付けない場合、lxc-start を実行したターミナルはコンテナ環境のコンソールに接続されます。

```
# lxc-start -n debstudy1
  または
# lxc-start -n debstudy1 -d
```

4.3.7 コンテナの終了:lxc-stop

lxc-stop コマンドでコンテナを終了します。コンテナを終了すると、init プログラムが起動中のデーモンを終了させ、init プログラム自身も終了します。

```
# lxc-stop -n debstudy1
```

4.3.8 コンテナのコンソールへ接続する:lxc-console

lxc-console コマンドでコンテナ環境のコンソールへ接続します。

“Ctrl+a q” の順にキー入力すると、コンソールから抜けることができます。

```
# lxc-console -n debstudy1

Connected to tty 1
Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter Ctrl+a itself

Debian GNU/Linux 8 debstudy1 tty1

debstudy1 login:
```

4.4 lxc を実用する

4.4.1 lxc のゲスト環境をを利用可能な状態までセットアップする流れ

lxc のゲスト環境へのログインは ssh でログインしたいと思うことが多いと考えます。そのため、lxc のゲスト環境を作成して動作させるまでのセットアップの流れの一例をまとめてみます。

なお、debian の場合は lxc-create したときの rootfs 内に openssh-server パッケージもインストールされます。

- lxc-create コマンドでゲスト環境を作成する
- config ファイルを修正し、IP アドレス付与及びネットワーク設定を行う
- chroot コマンドで rootfs へ直接入る (または、lxc-start でコンテナ環境を起動してコンソールへ接続する)
  - passwd コマンドで root パスワードを書き換える
  - adduser コマンドでユーザを作成する
  - apt-get install sudo vim-tiny
  - visudo
  - dpkg-reconfigure locales (初期値は lxc-create したときに LANG 値となります)
- lxc-start -n {lxc-name} -d を実行し、バックグラウンドでゲスト環境を起動
- ホスト環境から ssh コマンドでゲスト環境の IP アドレスとユーザを指定してログインする

ssh ログインができ、sudo が利用できるようになれば後はお好みで設定ができると思います。

4.4.2 何に lxc を使うか

lxc をどのような場面で利用するかはその人次第です。自分は以下の用途で利用しています。

- 一時的な検証で、ホスト環境にいろいろインストールしたくない場合 (例：デーモンの再起動が必要になる)
- python2 系と python3 系が混在した複数の web アプリケーションを 1 つのホストで動かしたい場合 (debian の場合、libapache2-mod-wsgi と libapache2-mod-wsgi-py3 は共存できない)
- ホスト環境は systemd のままで、ゲスト環境は sysvinit で動作させたい場合 (systemd に対応しないプログラムを利用する苦肉の策)
- 開発したアプリケーションをクリーンな環境でビルドやインストールできるかテストを行う場合[*8][*9]
- amd64 上で i386 環境がほしい、または異なる CPU アーキテクチャのコンテナ環境が動作するクロス環境がほしい場合[*10]

4.5 おわりに

Debian GNU/Linux 上で lxc を試してみました。lxc は、複数ホストの lxc 環境を制御する LXD や docker の基礎技術であるため、皆さま試してみてください。

4.6 参考文献

- 「LXC」 `https://linuxcontainers.org/`
- 「LXC - Debian Wiki」 `https://wiki.debian.org/LXC`
- 杉本 典充 (2013) 「debootstrap を有効活用してみよう」 `http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume201304.pdf`

[*8] きちんと debian パッケージにしてインストールするようにすれば、この用途で利用することはないはずです

[*9] debian にある pbuilder や cowbuilder はクリーンな rootfs へ chroot して debian パッケージのビルドができるかをテストする便利コマンドです

[*10] cross debootstrap といい、QEMU と組み合わせます。

Debian 勉強会資料

2016 年 8 月 20 日　　初版第 1 刷発行

東京エリア Debian 勉強会 （編集・印刷・発行）