目次

第 1 章

はじめに

hogehoge

第 2 章

KVM

2.1 KVM, QEMU

hogehoge

2.1.1 KVM, QEMU, OpenStack

hogehoge

- KVM
- OpenStack

1. KVM
2. Docker
3. LXC

```
./stack.sh
```

hogehoge

第 3 章

Containers 超入門

3.1 Containers の世界と LXC、そして Docker

3.1.1 昔からあるコンテナ技術

コンテナ技術を取り囲む現在の状況と、それを踏まえた上での LXC と Docker の根本的な違いについて説明したいと思う。Linux Containers(LXC) は、どうやって我々がアプリケーションを動かしスケールさせるかという問題を変化させる可能性を持っている。コンテナ技術は新しいものではない。そして、LXC に関して言うと、追加パッチを Linux Kernel に適用させることなく、vanilla Linux Kernel 上で稼働させることができる。なお、LXC の Version1 は、長期サポートバージョンであり、5 年間サポートされることになる。話が逸れるが、vanilla Linux Kernel とは、Linux 作者の Linus Torvalds 氏がリリースするプレーンな Kernel のことである。それをベースに様々なベンダーが追加で拡張していくのである。また、vanilla という言葉には「普通の、ありきたりな、おもしろみのない」という意味がある。

話を戻そう。

コンテナ技術は、最近登場した新技術ではない。昔から存在し色んな所で採用されている。FreeBSD には Jail があり、Solaris には Zone がある。それに加えて、OpenVZ や Linux VServer のような Containers も存在する。その歴史は、chroot に始まり、FreeBSD Jail を経て、Linux Containers(LXC) に至る。chroot では、大雑把に言って、ディレクトリーツリーの分離を行っていた。プロセスリスト自体は共有するようなモデルである。chroot のユースケースとしては、開発者向けのテスト/ビルド用環境である。FreeBSD Jail では、chroot の機能に加えて、プロセスリストとネットワークスタックも分離 (というか隔離) された。ユースケースとしては、root 権限の一般ユーザへの委譲、またそれに頼る形でのホスティングサービスである。LXC では、リソース管理テーブルを隔離し、cgroups によるシステムリソース (CPU、メモリ、ディスク etc) の制御を行えるようになった。これにより、LXC は、軽量な仮想環境と見なすことができるようになった。

3.1.2 なぜ皆コンテナに騒いでいるのか

コンテナは、ホストシステムからアプリケーションのワークロードを隔離、あるいはカプセル化する。コンテナを、ホスト OS 内にあるアプリケーションが実行されている OS と見なすことができ、かつ、それは Virtual Machine のように振る舞うのである。このエミュレーションは、Linux Kernel それ自体と、様々なディストリビューションとコンテナを使ってアプリケーションを動かすユーザのためにコンテナ用 OS のテンプレートを提供する LXC Project によって、実現されている。このように、Containers 技術が仮想マシンのように振る舞うことが可能になったことが、一気に注目を浴びる原因となったのである。

3.1.3 コンテナの価値は?

コンテナは、アプリケーションをホスト OS から分離、抽象化することで、LXC をまたぐシステム間でのポータビリティをもたらす。また、コンテナは、ハードウェアをエミュレートすることなく、cgroups と namespaces を駆使して Linux Kernel 内でベアメタルに近いスピードの軽量な OS 環境を実現している。シンプルで高速、かつハードウェアの仮想化よりもよりポータビリティーがありスケールしやすいという構造により、コンテナは、根本的なユーザのワークロードやアプリケーションの仮想化の方法を変えるものなのである。なお、ここでいうポータビリティーとは Docker によりもたらされる、どこでも同一のアプリケーションを稼働することができるという意味ではない。

3.1.4 LXC

LXC プロジェクトは、コンテナ用の OS テンプレートとライフサイクル管理のための幅広いツールセットを提供しています。現在、Canonical のサポートのもと、Stephane Graber と Serge Hallyn により開発は主導されています。

LXC は、活発に開発されているがその割にはドキュメントが少ない。特に Ubuntu 以外のディストリビューションで利用する際のドキュメントが欠如しており、多くの機能はまず Ubuntu 上で実装される。他のディストリビューションを利用しているユーザーからしたら、とてもフラストレーションのたまることである。また、ネット上には数多くの誤解を招くような情報があふれ、混乱を招いている状況も少なからずある。広くマーケットで存在感を示している Docker と混同されたり、そもそもの情報の多さが混乱の元となっていたりする。

LXC は、Docker のようなフロントエンドのアプリケーションのためのローレイヤー層なのか、はたまた、Docker が LXC 上に構築されたユーザーフレンドリーなフロントエンドなのか？こういった不確かな情報が広く出回っている。コンテナ技術のメリットを享受するために必ずしも Docker を使う必要はない。Docker はコンテナ利用の一つの選択でしかないのである。

3.1.5 Docker と LXC では何が違うのか

Docker 視点から見た LXC との違いについて説明する。

そもそも LXC に対して Docker が提供している機能とは何なのか。まず第一に言われることは、どのようなホスト OS であってもポータブルなデプロイが可能である点である。Docker はアプリケーションをビルドするためのフォーマットが定義されている。この定義をまとめて記述するために Dockerfile ファイルを利用する。この Dockerfile ファイルはビルドでよく使われる makefile ファイル同様に Docker Containers の構成情報をまとめて記述するテキストファイルである。このファイルに記述する定義情報が全ての依存関係をカプセル化しているため、それはどこで実行してもアプリケーション実行環境が同一になるのです。LXC のプロセスのサンドボックスもポータビリティーを持っているが、もし LXC のコンフィグをカスタマイズしているとしたら、ネットワークやストレージ、ディストリビューションの違いにより、それは別環境で稼働しない可能性が高くなります。Docker はその全てを抽象化するためどんな環境でも稼働させることができるのである。

Docker について言及するエンジニアは、総じてアプリケーション寄りのエンジニアである。Docker は、軽量な仮想マシンとしての利用というよりもアプリケーションのデプロイに最適化されている。これは Docker 自体の API やデザインの設計思想に反映されている。それとは対照的に、LXC は軽量な仮想マシンとしての利用に注力している。Docker には、git に似たバージョン管理機能が含まれている。バージョン間の diff の取得や Commit、ロールバックが可能となっている。それによって、Containers の変更を誰がどのように行ったのかに

ついての全てのログを追うことができる。

他にも Docker の利点は多く存在するが、主にこのような点により、Docker は、コンテナそのもの対する見方を変えるきっかけを作った。今まで軽量な仮想マシンとして見られていたコンテナ技術をアプリケーションとしてのコンテナとしてエンジニアに再認識させることに成功したのである。

3.2 LXC を使い軽量仮想環境を手に入れよう

LXC の基本的なコマンドを使ったコンテナ操作を，Ubuntu14.04 をベースにした環境を使って説明していきたい。

3.2.1 LXC のインストール

Ubuntu の最新版である Ubuntu 14.04 LTS では、LXC 1.0.7 が lxc というパッケージ名で提供されている。また、Debian 8 (Jessie) では、LXC 1.0.6 のパッケージが提供されている。

インストールは以下のコマンドを叩くだけである。

```
$ sudo apt-get install lxc
```

3.2.2 LXC で仮想環境を立ち上げる

LXC による仮想環境を立ち上げるためには、まずテンプレートと呼ばれる設定ファイルを用いる。デフォルトでメジャーディストリビューションのテンプレートはすでに同梱されているためこちらを利用する。テンプレートは/usr/share/lxc/templates に配置されている。

```
$ ls /usr/share/lxc/templates/
lxc-alpine      lxc-centos     lxc-fedora         lxc-oracle  lxc-ubuntu-cloud
lxc-altlinux    lxc-cirros     lxc-gentoo         lxc-plamo
lxc-archlinux   lxc-debian     lxc-openmandriva   lxc-sshd
lxc-busybox     lxc-download   lxc-opensuse       lxc-ubuntu
```

Ubuntu のテンプレートを用いて test-container-101 という名前の Ubuntu のコンテナを立ち上げる。

```
$ sudo lxc-create -t ubuntu -n test-container-101
```

これでコンテナの root ディレクトリに相当するディレクトリに必要なものがインストールされる。コンテナの場所は以下のディレクトリである。

```
/var/lib/lxcコンテナ名/<>/
```

そこで、test-container-101 の rootfs の中を覗いてみると以下の通りとなる。

```
$ ls -F /var/lib/lxc/test-container-101/rootfs/
bin/   dev/  home/  lib64/  mnt/  proc/  run/   srv/  tmp/  var/
boot/  etc/  lib/   media/  opt/  root/  sbin/  sys/  usr/
```

インストールしたコンテナの起動には以下のコマンドを実行する。

```
$ sudo lxc-start -n test-container-101 -d
```

-d オプションでデーモンとしてコンテナを起動する。この状態で lxc-console コマンドを用いてコンソール接続することでコンテナの内部にログインすることができる。

```
$ sudo lxc-console -n test-container-101
```

コンテナから抜ける際には、Ctrl+A を入力してその後 Q を押す。また、デフォルトの root パスワードは以下に保存されている。

```
/var/lib/lxc//tmp_root_pass
```

コンテナの終了は、lxc-shutdown コマンドを実行すればよい。

```
$ sudo lxc-shutdown -n test-container-101
```

3.2.3　コンテナネットワークの設定

lxcbr0 の図を入れる！！！！！！！！！！！！

3.2.4　コンテナの情報を見る

第 4 章

あとがき

4.1 こじろー

4.1.1 Container

hoge

piyo

hogehoge

4.2 まっきー

4.2.1 KVM

hoge

piyo

hogehoge

4.3 だーまり

hogehoge