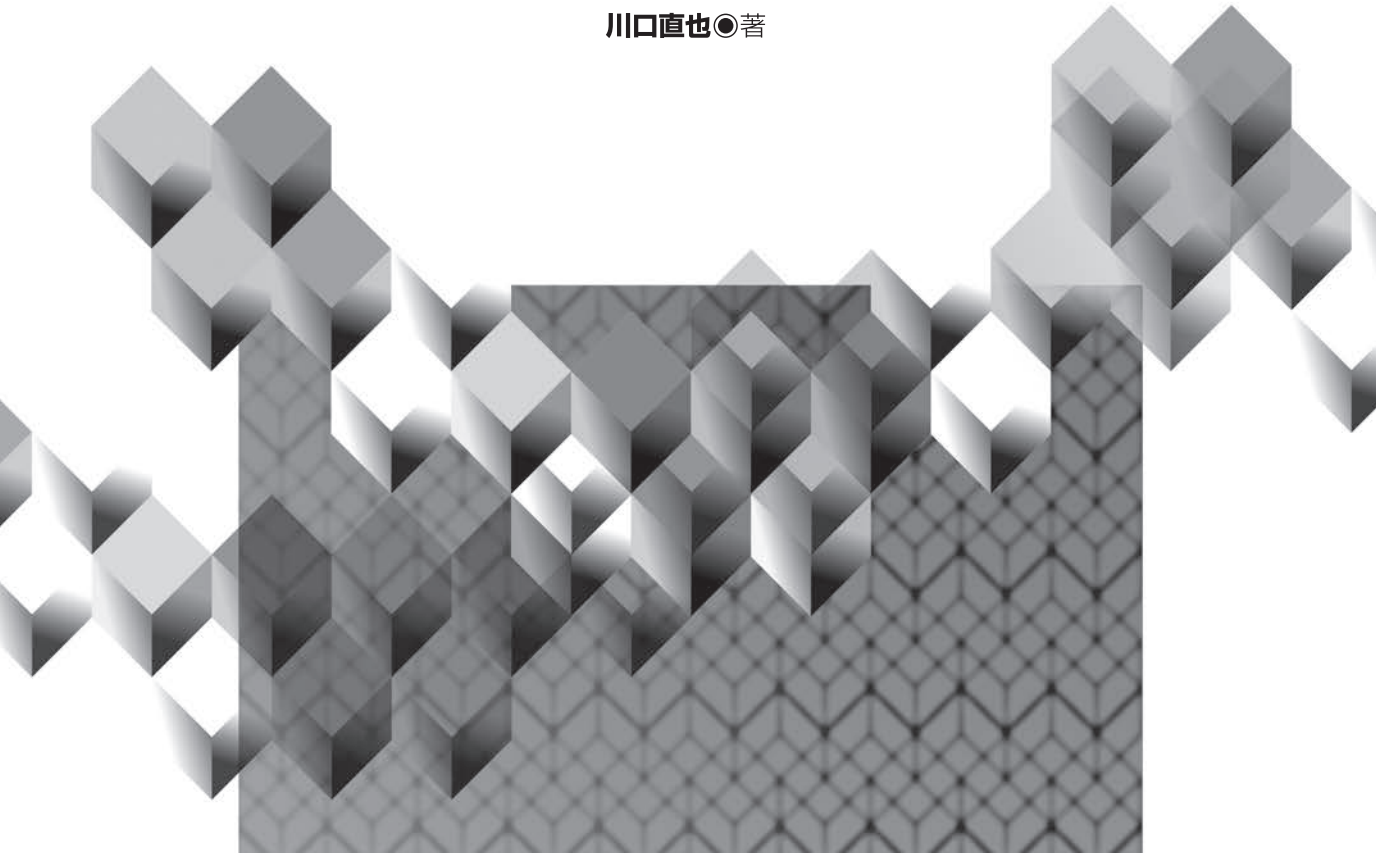
コンテナ型仮想化概論

川口直也◉著



CUTT
カットシステム

・本書の内容についてのご意見、ご質問は、お名前、ご連絡先を明記のうえ、小社出版部宛文書（郵送またはE-mail）でお送りください。
・電話によるお問い合わせはお受けできません。
・本書の解説範囲を越える内容のご質問や、本書の内容と無関係なご質問にはお答えできません。
・匿名のフリーメールアドレスからのお問い合わせには返信しかねます。

目次

第1章

はじめに

1.1 本書の概要

本書はコンテナ型仮想化技術の解説書です。コンテナ型仮想化とは、OS レベルで資源を仮想化することにより、ハードウェア仮想化に比べて軽量なアプリケーションの実行環境を提供する仕組みを指します。本書では、そのアーキテクチャや理論から実際の操作方法までを扱います。

第 1 章ではコンテナ型仮想化の概要や歴史について述べます。第 2 章ではコンテナ型仮想化の全般的な仕様について解説します。第 3 章から第 8 章では今日のメジャーなコンテナ型仮想化ソフトウェアの実際の操作方法について解説します。第 9 章では比較的マイナーではあるもののコンテナ型仮想化について知見を与えてくれる幾つかのコンテナ型仮想化ソフトウェアについて解説します。第 10 章ではコンテナ型仮想化ソフトウェアと近接したテーマであるサンドボックス環境などについて解説します。第 11 章では次世代のコンテナ型仮想化ソフトウェアについて紹介します。第 12 章では本書のまとめと今後のコンテナ型仮想化ソフトウェアの展望について言及します。

1.2　本書が想定する読者層

本書が想定する読者層は、コンテナ型仮想化ソフトウェアを導入する前段階として、そもそもコンテナ型仮想化技術とは何で、その歴史的変遷や今後の展望について把握する必要のあるユーザーです。

本書ではコンテナ型仮想化技術のうち、主に以下のトピックについて解説しています。

- コンテナ型仮想化技術とは何か
- コンテナ上では「何が」、「どこまで」できるのか
- どのようなコンテナ型仮想化ソフトウェアの選択肢があるのか
- 各コンテナ型仮想化ソフトウェアの実際の操作方法はどのように違うのか
- 従来のコンテナ型仮想化技術の歴史的変遷
- 今日のコンテナ型仮想化技術の課題と今後の展望

これらの知見を得ることにより、現在主流のコンテナ型仮想化ソフトウェアを漫然と受け入れて導入するのではなく、今日までのコンテナ型仮想化技術の歴史的経緯を踏まえた上で、今後を見越した導入計画の策定や運用方法の見直しを行うことができるでしょう。

本書ではコンテナ型仮想化技術の俯瞰的な知識の習得を目的としているため、以下の用途には本書は適していないか、少なくとも本書を単独で用いることは望ましくありません。

- Docker などの個別のコンテナ型仮想化ソフトウェアの使用方法の純粋な習得
- 特定のコンテナ型仮想化技術にのみ特化した具体的な導入プランの策定
- Docker の高度な利用方法や導入事例の把握
- Kubernetes やクラウドサービスなど、コンテナの高水準管理システムの利用方法の習得

これらの知見が必要な方は、別途 Docker などの特定のソフトウェアの解説書を参照してください。

1.3 仮想化について

仮想化とは実在する一つの資源（リソース）を、あたかも複数個存在するかのように見せる技術です。例えば、仮想メモリは実際には一つしかない物理メモリ空間を仮想化して、複数のプロセスに対してあたかも独占的にメモリ空間を使用しているかのように見せかけています。また、仮想 NIC デバイスは NIC（Network Interface Card）を仮想化し、VPN や仮想スイッチの構築に用いられます。以下では、仮想化技術のうち、特にアプリケーションの実行環境を仮想化する VM とコンテナ型仮想化について解説します。

1.3.1 仮想マシンについて

仮想マシン（VM）は、一つの計算機全体を仮想化し、その上で動作する OS やアプリケーションに対してあたかも実機で動作しているかのように見せかける機構です。VM を用いることで単一の計算機を複数のユーザーに対して疎結合な状態で提供することができます。また VM 上で動作する OS やミドルウェア、アプリケーションによる障害が他の VM 上で動作するアプリケーションに波及するリスクを大幅に低減することができます。

以下に主要な VM の実装を紹介します。

- Hyper-V
- VMware
- KVM
- Xen
- Virtual Box
- QEMU
- Bochs

上記の VM のうち、QEMU と Bochs を除いた VM では、ハイパーバイザという技術が取り入れられています。ハイパーバイザは VM 上のアプリケーションが実行する特権命令以外の機械語命令を実機の CPU が解釈して実行する仕組みです。最近の x86 系 CPU ではハイパーバイザのオーバーヘッドを低減するために、Intel-VTd というハードウェアレベルでの仮想化支援機構が取り入れられています。

ハイパーバイザは VM を比較的低いオーバーヘッドで実現することができますが、一方で仮想化する CPU アーキテクチャとハイパーバイザが動作する環境のアーキテクチャが同一である必要があります。

これに対し、QEMU や Bochs は CPU の命令列をエミュレーションする方式を採用しています。このため、ターゲットの CPU アーキテクチャとは別のアーキテクチャ上で VM を実行することができます。

仮想マシンの歴史は古く、1965 年にリリースされた IBM 製メインフレームの System/360 には既に仮想マシンが実装されていました [1]。System/360 では複数のユーザーがあたかも一つのメインフレームを占有しているかのように操作することが可能でした。その後、仮想マシンは高価なメインフレームに実装された技術として育まれていきます。

この状況に対して、2000 年代初頭に転機が訪れます。x86 ベースのアーキテクチャ上で動作する VMware や Xen といった仮想マシンが登場し、流行し始めます。そして、クラウドコンピューティングのブームと相まって、仮想マシンはデータセンターにおいて重要な役割を担うことになります。

1.3.2　コンテナ型仮想化について

コンテナ型仮想化（または OS レベル仮想化）とは、仮想マシン（VM）によるハードウェアレベルでの仮想化とは異なり、OS レベルでの仮想化（コンテナ）を目指す試みです。コンテナ型仮想化では、一つのハードウェアの上であたかも複数の OS（コンテナ）が動作しているかのように見せかけることができます。コンテナ型仮想化では、計算機のハードウェア全体を仮想化する VM に対して、OS レベルでの仮想化を実現するため、比較的軽量に実装できます。この仕組みを利用すると、低オーバーヘッドで、異なるコンテナ上で動作するアプリケーション間の影響を低減した運用が可能です。

■ コンテナ型仮想化の歴史

ここではコンテナ型仮想化の歴史について解説します。コンテナ型仮想化の起源は、1970 年代に登場した UNIX の `chroot` システムコールに遡ります [2][3]。`chroot` システムコールは、プログラムがファイルツリーの部分木を構成するファイル群にのみアクセスできるようにするシステムコールです。`chroot` システムコールを使用することで、アプリケーションが利用できるファイル空間を限定することができるようになりました。

1990 年代後半には IBM の AIX v6.1 から WPAR（Workload Partition）[4] が搭載され、ヒュー

レットパッカードの HP-UX もコンテナを実装しました [5]。これらの商用 UNIX は PowerPC や Itanium といった x86 系ではないプロセッサ上で動作する OS のため、一般のユーザーがコンテナの恩恵を受けることは、まだそれほどありませんでした。

その後、2000 年に FreeBSD jail[6] が登場します。FreeBSD jail は UNIX の root ユーザー権限を維持したままで OS 環境を分離することができます。これにより一般ユーザーに対してコンテナ内でのみ利用できる root 権限を付与することができるようになりました。以降に登場するコンテナ型仮想化技術は基本的に FreeBSD jail の仕様を踏襲しています。

2005 年には Solaris ゾーン [7] が登場します。Solaris ゾーンはゾーンと呼ばれる区画に仮想の OS 環境を構築することができます。ゾーンには Solaris マシンに一つだけ存在するグローバルゾーンと複数個の仮想環境を構築できる非グローバルゾーンがあります。グローバルゾーンは非グローバルゾーンを管理することができます。システム全体をゾーンという枠組みで捉えなおし、グローバルゾーンから非グローバルゾーンを管理するという考え方が Solaris ゾーンの特徴です。

2006 年には namespace 機能が Linux カーネル 2.6 系列に実装されます。namespace 機能はプロセスが所属する名前空間を分離することにより、各プロセスが独自の空間で動作しているかのように見せかける機能です。例えば、ps コマンドによって見えるプロセス空間を分離することにより、関係のないプロセス群をユーザーに対して隠蔽することができます。この namespace 機能を利用して登場したのが LXC[8] や Docker[9] です。

1.4 参考文献

[1] 仮想化の基本と技術（仕組みが見えるゼロからわかる）, 清野 克行 , ISBN-13:978-4798123707

[2] いまさら聞けないコンテナの歴史 「chroot」から「Docker」まで：技術の変遷を解説 - TechTarget ジャパン システム開発 , https://techtarget.itmedia.co.jp/tt/news/1807/26/news06.html

[3] Man page of CHROOT, https://linuxjm.osdn.jp/html/LDP_man-pages/man2/chroot.2.html

[4] IBM Workload Partitions for AIX, https://www.ibm.com/support/knowledgecenter/ja/ssw_aix_71/workloadpartitions/wpar-kickoff.html

[5] HP-UX Containers (SRP) A.03.01 Administrator's Guide, https://support.hpe.com/hpesc/public/docDisplay?docId=emr_na-c02918260
[6] Jails: Confining the omnipotent root., http://phk.freebsd.dk/pubs/sane2000-jail.pdf
[7] Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture, Richard Mauro, Jim McDougall et al, ISBN-13:978-0131482098
[8] Containerization with LXC, Konstantin Ivanov, ISBN-13:978-1785888946
[9] Docker, Adrian Mouat（著）, Sky 株式会社 玉川 竜司（翻訳）, オライリージャパン , ISBN-13:978-4873117768

第2章

コンテナ型仮想化の概要

2.1 コンテナ型仮想化の特徴

コンテナ型仮想化は、OS 層を仮想化することで、あたかも複数の OS が動作しているかのように振る舞う技術です。コンテナ型仮想化を用いることで、アプリケーションに対して OS 資源を占有しているかのように見せかけることができます。また、ユーザーに対しても仮想化した OS の root 権限を付与することができるため、個々のアプリケーションの運用における柔軟性や利便性を高め、ひいてはシステム全体の利用効率や集積性を高めることができます。こうした仮想化技術の恩恵は仮想マシンを用いても享受することができます。しかし、コンテナ型仮想化ではこれらの恩恵を仮想マシンに比べて低オーバーヘッドで実現できるという特性があります。

一方でコンテナ型仮想化では、仮想化する対象の OS は、基本的にコンテナの下層で動作している OS と同一のアーキテクチャのものに限定されます。例えば、Solaris ゾーンの上で Linux や Windows といった異種 OS のアプリケーションを実行することはできません。

また、コンテナ型仮想化では、コンテナに対して物理的なハードウェアを割り当てる仕組みが仮想マシンほど整っておらず、キャパシティプランニングが難しいといった特徴があります。特に、あるコンテナ上で著しくリソースを消費するアプリケーションが実行されている場合に、別のコンテナに対してその影響が波及する可能性があります。こうした特性があるためにパブリッククラウドのような、多種多様なユーザーやアプリケーションの実行に供する用途

では慎重な導入が求められます。

コンテナ型仮想化では、ホストとなるシステム（ホスト OS）とコンテナ上のシステムとのやり取りを簡単に行うことができます。例えば、コンテナの内外でファイル共有を行う場合が挙げられます。また、ホスト OS のコンソールからコンテナ内のコンソールに動的にアタッチして操作することができます。このようにコンテナ型仮想化ではホスト OS とコンテナ上のゲストシステムとの結合度が、仮想マシンに比べて高いという特性があります。

このほか、コンテナ型仮想化ではコンテナの生成や破棄、起動や終了といった操作を VM に比べて高速に実行することが可能です。

2.2　コンテナ型仮想化のアーキテクチャ

コンテナ型仮想化の多くは、ホスト OS のカーネルの一部機能として実装されています。

例えば、FreeBSD Jail では jail システムコール [1] を用いて、ホスト OS のユーザーモードプログラムからコンテナを管理することができます。

Solaris ゾーンでは、ホスト OS が動作する領域をグローバルゾーンと呼称し、コンテナ上のアプリケーションが動作する領域を非グローバルゾーンと称します。グローバルゾーンから非グローバルゾーンを操作するには、Solaris 環境にデフォルトで用意されている `zoneadm` コマンド [2] を使用します。

Linux の場合も、コンテナ型仮想化を実現するための namespace 機能や CGroups 機能がカーネルに組み込まれています。

このようにコンテナ型仮想化では OS 層を仮想化するために、OS 層において特別な実装が施されています。

■ コンテナの管理用インターフェース

コンテナの作成や開始、停止といった操作は、コマンドにより行う方式が一般的ですが、FreeBSD Jail のように専用のシステムコールが用意されているものもあります。また、LXC では通常のコマンド以外にも LXD という上位システムが提供するコマンド群があります。Docker は通常は docker コマンドによりコンテナを管理しますが、CRI というインターフェースも定義されており、Kubernetes のような上位システムからプログラム的にコンテナを管理することもできます。

●コンテナの管理用インターフェース

コンテナ	操作方法
FreeBSD Jail	・service コマンドや jexec コマンドなど ・jail システムコール
Solaris ゾーン	zoneadm コマンド
OpenVZ	vzctl コマンド
LXC	・lxc-XXX コマンド（例）lxc-start、lxc-stop ・lxd コマンド
Docker	・docker コマンド ・CRI（Container Runtime Interface）

■ ストレージ領域

コンテナから利用できるストレージ領域は、通常はホスト OS の通常のファイルシステム空間の一部を chroot 化して利用します。この時、chroot 化する領域にはコンテナのルートファイルシステムが存在します。コンテナのルートファイルシステムには /bin や /home といったファイル群の他にも、通常は /dev や /proc といった特殊なファイルシステムが含まれます。

また、Docker にはボリュームというコンテナランタイムが管理する領域が存在し、ボリュームを経由して、複数のコンテナ間でファイルを共有することもできます。

Solaris ゾーンでは ZFS を利用して、chroot 化するコンテナのルートファイルシステムを、ZFS 上の独立した単位として扱うことにより、コンテナ単独でのスナップショットの取得ができます。

■ コンテナ間の隔離

chroot システムコールによりコンテナを構成するファイル群は隔離することができますが、プロセス空間やネットワークインターフェースなども別途隔離する必要があります。Linux の namespace 機能では、コンテナ間で以下の名前空間を隔離することができます。

● Linux の namespace 機能で隔離できる名前空間

名前空間	説明
IPC 名前空間	セマフォや共有メモリが所属する名前空間
マウント名前空間	プロセスから見えるマウントポイントの一覧を隔離する。 例えば、ホスト OS の /proc/[pid]/mounts ファイルには当該プロセスがマウントしているファイルシステムの一覧が表示される。 当該プロセスからは他のプロセスのマウント状況を見ることはできない。
ネットワーク名前空間	NIC やルーティングテーブルなどを隔離する。
PID 名前空間	プロセス ID が所属する名前空間を分離する。
ユーザー名前空間	UID や GID を隔離する。
UTS 名前空間	ドメイン名やホスト名を分離する。

2.3　コンテナ型仮想化の用途

コンテナ型仮想化を利用する主要なモチベーションは、低オーバーヘッドで OS レベルの仮想環境を提供し、アプリケーションが利用できる特権的な環境を整備することにあります。OS レベルで仮想環境を提供することにより、アプリケーション群に対して、プロセスレベルでの分離性では足りないが、マシンレベルでの分離性は不要というケースに対応できます。

2.4　参考文献

[1]　jail システムコール , https://www.freebsd.org/cgi/man.cgi?query=jail

[2]　zoneadm(1M) - 形式 - マニュアルページセクション 1M ..., https://docs.oracle.com/cd/E26924_01/html/E29114/zoneadm-1m.html

第3章

FreeBSD Jail

3.1 FreeBSD Jailとは

FreeBSD Jail[1][2] は、FreeBSD に実装されたコンテナ型仮想化機構です。FreeBSD Jail は2000 年に登場した FreeBSD 7.2 からサポートされており、比較的歴史のあるコンテナ型仮想化機構です。このため、ホスティングサービスなどで多くの採用実績があります。

FreeBSD Jail は次章で紹介する Solaris ゾーンとは異なり、コマンドのほかにシステムコールレベルでコンテナを操作する機能を公開しています。これは、任意のアプリケーションから動的にコンテナを操作できることを意味します。

FreeBSD Jail を用いたコンテナの作成では、まずコンテナ内に配置するファイル群であるルートファイルシステム（/bin など）をユーザーが手動で作成します。これは他のコンテナ型仮想化ソフトウェアと大きく異なる点です。

FreeBSD Jail において、コンテナは Jail と呼びます。Jail は FreeBSD の特定のバージョンに紐づいており、そのバージョンは後述するベースシステムによって Jail の構築時に設定することができます。例えば、FreeBSD 12.1 の Jail と 9.3 の Jail を同一マシン上に共存させることができます。このように、複数の OS バージョンのコンテナが共存できることも、FreeBSD Jail の特徴の一つです。

Jail の構築や利用に際しては、幾つかの方法が用意されており、ユーザー自身で jail のすべ

ての設定や起動を行う方法と、ezjail や iocage といったツールを利用して jail の設定や起動を行う方法があります。

3.2　FreeBSD のインストール

本節では FreeBSD のインストールについて解説します。本書では以下のバージョンの FreeBSD を使用して解説を行います。

バージョン	FreeBSD-12.1-RELEASE
プラットフォーム	amd64
ISO ファイル名	`FreeBSD-12.1-RELEASE-amd64-dvd1.iso`
配布元	Download FreeBSD `https://www.freebsd.org/where.html`

FreeBSD のインストール作業は基本的にデフォルト設定のままで構いません。

また、FreeBSD Jail は Intel-VT などの仮想マシン支援機構を用いないため、Hyper-V などの仮想マシン上に FreeBSD を構築しても、Jail を利用することができます。筆者は本書の執筆にあたり、Hyper-V の仮想マシン上に FreeBSD を構築しました。

3.3　FreeBSD の設定

FreeBSD のインストールが完了したら、IP アドレスの設定や SSH サーバーの設定といった最低限の作業を行う必要があります。Linux と違い、FreeBSD では IP アドレスはデフォルトで DHCP に設定されておらず、また SSH もデフォルトでは無効化されています。

なお、FreeBSD はデフォルトでは GUI を利用できないため、Linux（Ubuntu）のように GNOME から IP アドレス設定を行うような操作はできません。このため、設定はすべてコンソール上でエディタを使用して行うことになります。

3.3.1 FreeBSD の IP アドレス設定

FreeBSD の IP アドレスは以下のファイルを編集して設定します。

```
/etc/rc.conf
```

IP アドレスを DHCP に設定する場合は、/etc/rc.conf ファイルを以下のように編集してください。

DHCP の設定例

```
ifconfig_hn0="DHCP"
```

上記の設定例 hn0 は NIC のデバイス名です。NIC のデバイス名称は環境によって異なる場合があるため、ifconfig コマンドで確認してください。

IP アドレスを固定で設定する場合は、/etc/rc.conf ファイルを以下のように編集してください。

固定 IP の設定例

```
ifconfig_hn0="inet 192.168.0.2 netmask 255.255.255.0"
defaultrouter="192.168.0.1"
```

上記の例では、192.168.0.2 の IP アドレスを hn0 の NIC に設定し、デフォルトゲートウェイを 192.168.0.1 に設定しています。

3.3.2 FreeBSD の SSH 設定

FreeBSD で SSH を利用できるようにするには、以下の二つのファイルを編集します。

- /etc/rc.conf
- /etc/ssh/sshd_config

/etc/rc.conf ファイルに以下の行を追加してください。

```
sshd_enable="YES"
```

/etc/ssh/sshd_config ファイルの以下の行のコメントアウトを解除し、必要に応じて編集してください。

```
Port 22
Protocol 2
PasswordAuthentication yes
PermitRootLogin no
PermitEmptyPasswords no
```

/etc/ssh/ssh_config という似た名前のファイルがあるため注意してください。

上記の設定が終わったら、以下のコマンドで SSH サーバーを再起動します。

```
# service sshd restart
```

SSH サーバーの再起動が完了したら、TeraTerm などのクライアントから SSH で FreeBSD に接続を行ってください。

3.3.3　Jail の有効化

/etc/rc.conf ファイルに以下の行を追加して、FreeBSD の Jail 機能を有効化します。

```
jail_enable="YES"
```

3.3.4　vim のインストール

本手順は必須ではありませんが、ここまでの設定で vi を使用して FreeBSD の設定を行ってきましたが、vim をインストールして使用することもできます。

vim のインストールは以下のコマンドで行います。

```
# pkg update
# pkg install vim
```