



# Citrix XenServer® 6.5 ソフトウェア開発キットガイド

発行 2015/03/26  
1.0 版



Citrix XenServer® 6.5 ソフトウェア開発キットガイド

Copyright © 2015 Citrix All Rights Reserved.

Version : 6.5

Citrix, Inc.  
851 West Cypress Creek Road  
Fort Lauderdale, FL 33309  
United States of America

#### 免責

このドキュメントは現状有姿のまま提供されます。Citrix, Inc.は、このドキュメントの内容に関し、商品性および特定目的適合性についての黙示保証を含むが、それに限定することなく、いかなる保証も行わないものとします。このドキュメントには、技術的に不正確な記述または印字エラーが含まれている可能性があります。Citrix, Inc.は、このドキュメントに含まれている情報を予告なく随時変更する権利を留保します。このドキュメントおよびこのドキュメントに記載されているソフトウェアは、Citrix, Inc.およびそのライセンス付与者の機密情報であり、Citrix, Inc.によるライセンス許諾に基づいて提供されます。

Citrix Systems, Inc.、Citrix ロゴ、Citrix XenServer、および Citrix XenCenter は、米国およびその他の国における Citrix Systems, Inc.の商標です。このドキュメントに記載されているその他のすべての製品またはサービスは、該当する各社の商標または登録商標です。

#### 商標

Citrix ®

XenServer ®

XenCenter ®

## 目次

<b>1. はじめに .....</b>	<b>1</b>
<b>2. 導入 .....</b>	<b>2</b>
2.1. システム要件と準備 .....	2
2.2. ダウンロード .....	2
2.3. 新機能 .....	2
2.4. 構成内容 .....	2
<b>3. XenServer API の概要 .....</b>	<b>4</b>
3.1. API の使用方法 .....	4
3.1.1. 認証：セッションリファレンスの取得 .....	4
3.1.2. 新しい仮想マシンのインストールで使用するテンプレートの一覧の取得 .....	5
3.1.3. テンプレートからの仮想マシンのインストール .....	5
3.1.4. 仮想マシンの起動/一時停止/再開/停止のサイクル .....	5
3.1.5. ログアウト .....	6
3.1.6. インストールと起動の例：概要 .....	6
3.2. オブジェクトモデルの概要 .....	6
3.3. 仮想ネットワークインターフェイスおよび仮想ブロックデバイスの操作 .....	9
3.3.1. ディスクの作成と仮想マシンへの接続 .....	9
3.3.1.1. 新しい空のディスクイメージの作成 .....	9
3.3.1.2. ディスクイメージの仮想マシンへの接続 .....	10
3.3.1.3. 仮想ブロックデバイスのホットプラグ .....	11
3.3.2. ネットワークデバイスの作成と仮想マシンへの接続 .....	11
3.3.3. ネットワークとストレージのためのホストの設定 .....	12
3.3.3.1. ホストのストレージ設定：PBD .....	12
3.3.3.2. ホストのネットワーク設定：PIF .....	12
3.4. 仮想マシンのエクスポートとインポート .....	12
3.4.1. Xen 仮想アプライアンス (XVA) 仮想マシンのインポート形式 .....	14
3.5. XML-RPC に関する注意事項 .....	17
3.5.1. datetime .....	17

3.6. 参考資料 .....	17
<b>4. API の使用 .....</b>	<b>18</b>
4.1. 典型的なアプリケーションの構造 .....	18
4.1.1. 低レベルトランスポートの選択 .....	18
4.1.2. 認証とセッションの処理 .....	19
4.1.3. オブジェクトリファレンスの検索 .....	19
4.1.4. オブジェクトに対する同期操作の呼び出し .....	20
4.1.5. タスクを使用した非同期操作の管理 .....	20
4.1.6. イベントのサブスクライブおよびリッスン .....	20
4.2. 言語バインディング .....	21
4.2.1. C .....	21
4.2.2. C# .....	22
4.2.3. Java .....	22
4.2.4. PowerShell .....	23
4.2.5. Python .....	24
4.2.6. コマンドラインインターフェイス (CLI) .....	24
4.3. 完全なアプリケーションの例 .....	25
4.3.1. XenMotion による複数の仮想マシンの同時移行 .....	25
4.3.2. xe CLI による仮想マシンの複製 .....	27
<b>5. HTTP による XenServer との対話 .....</b>	<b>29</b>
5.1. 仮想マシンのインポートとエクスポート .....	29
5.2. XenServer パフォーマンス統計値の取得 .....	29
<b>6. XenServer API 拡張 .....</b>	<b>31</b>
6.1. 仮想マシンコンソールの転送 .....	31
6.1.1. API による VNC コンソールの取得 .....	32
6.1.2. Linux 仮想マシンの VNC 転送の無効化 .....	32
6.2. 準仮想化 Linux のインストール .....	33
6.2.1. Red Hat Enterprise Linux 4.1/4.4 .....	33
6.2.2. Red Hat Enterprise Linux 4.5/5.0 .....	33
6.2.3. SUSE Linux Enterprise 10 Service Pack 1 .....	34

6.2.4. CentOS 4.5/5.0 .....	34
6.3. Xenstore エントリの仮想マシンへの追加 .....	34
6.4. セキュリティの強化 .....	34
6.5. ネットワークインターフェイスの詳細設定 .....	35
6.5.1. ethtool 設定 .....	35
6.5.2. そのほかの設定 .....	36
6.6. ストレージリポジトリ名の国際化対応 .....	36
6.7. XenCenter でのオブジェクトの非表示化 .....	37
<b>7. XenCenter API 拡張 .....</b>	<b>38</b>
7.1. pool .....	38
7.2. Host .....	38
7.3. VM .....	39
7.4. SR .....	41
7.5. VDI .....	41
7.6. VBD .....	42
7.7. network .....	42
7.8. VM_guest_metrics .....	42
7.9. Task .....	42



# 第 1 章 はじめに

XenServer の開発者ガイドへようこそ。このドキュメントには、XenServer が提供する SDK (Software Development Kit : ソフトウェア開発キット) を理解し、使いこなすために必要な情報が記載されています。この情報から、アーキテクチャの背景知識、基本となる API、提供されているツール、迅速に開発を軌道に乗せる方法を理解できます。

## 第2章 導入

XenServer には XML-RPC ベースの API が用意されており、より多くの XenServer 管理機能やツールを利用できます。XenServer API はローカルの XenServer ホストだけでなく、リモートシステムからもコールできます。未加工の XML-RPC コールで直接 XenServer の管理 API を使用するアプリケーションを作成することも可能ですが、各 API コールを主要プログラミング言語の第一級関数として使用可能にする言語バインディングを使用すると、アプリケーションをより簡単に開発できるようになります。XenServer SDK では、C、C#、Java、Python、および PowerShell 用の言語バインディングとサンプルコードが提供されます。

### 2.1. システム要件と準備

SDK を使用するには、まず XenServer をインストールする必要があります。無償版の Citrix XenServer は、<http://www.citrix.com/downloads/xenserver/> からダウンロードできます。インストールについて詳しくは、『XenServer インストールガイド』を参照してください。インストールが完了したら、ホストの IP アドレスとホストのパスワードを書き留めておきます。

### 2.2. ダウンロード

この SDK は ZIP ファイルとしてパッケージ化されており、<http://www.citrix.com/downloads/xenserver/> から無償でダウンロードできます。

### 2.3. 新機能

XenServer 6.5 SDK では、C、C#、Java、PowerShell、および Python 用の言語バインディングによる API が提供されます。このリリースには、以下のようないくつかの変更および拡張が施されています。

- **C、C#、Java、および PowerShell 用のバインディング**  
XenServer 4.0 のサポートは削除されています。
- **PowerShell バインディング**  
このリリースに含まれている PowerShell 言語バインディングには、多くのバグ修正が施された従来の XenServer PowerShell v1.0 のスナップインと、新たに再設計された XenServer PowerShell v2.0 スナップインの 2 つのバージョンがあります。詳しくは、『XenServer リリースノート』を参照してください。

### 2.4. 構成内容

SDK の ZIP ファイルの内容は、XenServer-SDK ディレクトリに含まれています。このディレクトリに、以下の内容が抽出されます。一部のサブディレクトリには、個別の README ファイルが収録されています。提供されるサンプルコードは、言語バインディングにより異なる場合があります。このため、ほかのバインディング用のサンプルコードも参照することをお勧めします。

- **XenServer-SDK**
  - **libxenserver**  
C 言語用の XenServer SDK。
  - **bin**  
libxenserver のバイナリ。
  - **src**  
libxenserver のソースコードとサンプル、およびこれらをビルドするためのメイクファイル。
- **XenServer.NET**  
C#.NET 言語用の XenServer SDK。



- **bin**  
XenServer.NET のバイナリ。
- **samples**  
XenServer.NET のサンプル。
- **src**  
XenServer.NET のソースコード。
- **XenServerJava**  
Java 言語用の XenServer SDK。
  - **bin**  
Java のバイナリ。
  - **javadoc**  
Java のドキュメント。
  - **samples**  
Java のサンプル。
  - **src**  
Java のソースコード、およびソースコードとサンプルをビルドするためのメイクファイル。
- **XenServerPSSnapIn**  
新たに再設計された PowerShell 言語用の XenServer SDK。このディレクトリには、XenServer PowerShell Windows インストーラ *XenServerPSSnapIn.msi* が格納されています。
  - **samples**  
PowerShell のサンプルスクリプト。
  - **src**  
XenServer PowerShell のソースコード。
- **XenServerPSSnapIn\_old**  
従来の PowerShell 言語用の XenServer SDK。このディレクトリには、XenServer PowerShell Windows インストーラ *XenServerPSSnapIn.msi* が格納されています。
  - **samples**  
PowerShell のサンプルスクリプト。
  - **src**  
XenServer PowerShell のソースコード。
- **XenServerPython**  
このディレクトリには、XenServer Python モジュール *XenAPI.py* が格納されています。ライブラリ *provision.py* は、サンプルコードで使用されます。
  - **samples**  
Python を使用した XenServer API のサンプル。



## 第3章 XenServer API の概要

この章では、XenServer API（以下、「API」と呼びます）および関連付けするオブジェクトモデルについて紹介します。API の主な機能は次のとおりです。

- **XenServer ホストの管理**  
API を使用して、仮想マシン、ストレージ、ネットワーク、ホストの設定、およびプールを管理できます。パフォーマンスと状態のメトリクスも API でクエリできます。
- **永続オブジェクトモデル**  
副次的な影響を持つすべての操作（オブジェクトの作成、削除、パラメータの変更など）の結果は、XenServer システムによって管理されるサーバー側のデータベースで保持されます。
- **イベントメカニズム**  
永続（サーバー側）オブジェクトが変更されたときに API を通じて通知されるように、クライアントを登録することができます。これによって、同時に実行するクライアントによるデータモデルの変更をアプリケーションで追跡できるようになります。
- **同期および非同期の呼び出し**  
すべての API コールを同期的に呼び出す、つまり完了までブロックすることができます。また、実行に時間がかかる可能性のある API コールを非同期的に呼び出すこともできます。非同期コールは、タスクオブジェクトへのリファレンスを即座に返します。このタスクオブジェクトの進行状況や状態情報について、API からクエリすることができます。非同期に呼び出された操作が完了すると、タスクオブジェクトから結果（またはエラーコード）を取得できます。
- **リモート実行が可能でクロスプラットフォームな API コール**  
API コールを発行するクライアントは、管理対象のホストに常駐している必要はありません。また、API を実行するために ssh でホストに接続する必要もありません。API コールは、XML-RPC プロトコルを使用してネットワーク経由で要求と応答を伝送します。
- **安全かつ認証されたアクセス**  
ホスト上で実行する XML-RPC API サーバーは、セキュアソケット接続を受け付けます。このため、クライアントは HTTPS プロトコルで API を実行できます。さらに、すべての API コールは、サーバーでユーザー名とパスワードを確認した上で確立されるログインセッションのコンテキストで実行します。これにより、XenServer への安全かつ認証されたアクセスが提供されます。

### 3.1. API の使用方法

API への手引きとして、まず XenServer 上で新しい仮想マシンを作成するために必要なコールと、仮想マシンの開始/一時停止/再開/停止サイクルについて説明します。ここでは特定の言語のコードには言及しません。この段階では、「インストールおよび開始」のタスクを遂行する RPC 呼び出しの順序について簡単に説明します。

#### 3.1.1. 認証 : セッションリファレンスの取得

まず初めに、`Session.login_with_password(<username>, <password>, <client_API_version>, <originator>)` をコールします。API はセッションベースのため、ユーザーはほかのコールを実行する前に、サーバーで認証されている必要があります。ユーザー名とパスワードが正しく認証された場合、このコールの結果がセッションリファレンスになります。これ以降の API コールでは、パラメータとしてセッションリファレンスを指定します。これにより、正しく認証された API ユーザーのみが XenServer 上で操作を実行できるようになります。すべての API コールで同じセッションを使用できます。セッションが完了したら、`Session.logout(session)` をコールして削除することをお勧めします。以降を参照してください。

### 3.1.2. 新しい仮想マシンのインストールで使用するテンプレートの一覧の取得

次に、ホスト上の「テンプレート」一覧をクエリします。テンプレートは特別な仮想マシンオブジェクトで、さまざまなゲストオペレーティングシステムに適したデフォルトパラメータが指定されています (`xe template-list` コマンドを実行すると、XenServer 上にインストールされているすべてのテンプレートを確認できます)。API でテンプレート一覧を取得するには、`is_a_template` フィールドが `true` に設定されている仮想マシンオブジェクトをサーバーで検索します。これを行うには、`VM.get_all_records(session)` をコールします。ここで `session` パラメータには、`Session.login_with_password` コールで取得したセッションリファレンスを指定します。このコールによりサーバーがクエリされ、すべての仮想マシンオブジェクトリファレンスとそれらのフィールド値のスナップショットが返されます。

(この段階では、返されたオブジェクトリファレンスとフィールド値を特定のクライアント言語で操作することを想定していません。詳細については、言語固有の API バインディングの章で具体的に説明します。ここでは、API コールによって返されたオブジェクトとフィールド値を読み取って操作するためのメカニズムが存在することを理解しておいてください)

すべての仮想マシンオブジェクトのフィールド値のスナップショットをクライアントアプリケーションのメモリ内に取得できたので、これをさらに反復処理し、`is_a_template` フィールドが `true` の仮想マシンオブジェクトを検索できます。次に、このクライアントアプリケーションでさらにテンプレートオブジェクトを反復処理して、`name_label` フィールドの値が「Debian Etch 4.0」(XenServer のデフォルトの Linux テンプレートの 1 つ) であるリファレンスを取得します。

### 3.1.3. テンプレートからの仮想マシンのインストール

引き続き同じ例を使用して説明します。前の手順で選択した「Debian Etch 4.0」テンプレートを使って、新しい仮想マシンをインストールします。このインストールには、2 つの API コールが必要です。

- 最初に、API コール `VM.clone(session, t_ref, "my first VM")` を使用します。これにより、新しい仮想マシンオブジェクトを作成するために、`t_ref` で参照される仮想マシンオブジェクトが複製されます。このコールの戻り値は、作成された仮想マシンの仮想マシンリファレンスです。これを `new_vm_ref` とします。
- この段階で、`new_vm_ref` が参照しているオブジェクトは、複製元の `t_ref` が参照している仮想マシンオブジェクトと同様に、まだテンプレートの状態です。`new_vm_ref` を仮想マシンオブジェクトにするには、`VM.provision(session, new_vm_ref)` をコールします。このコールが戻ると、`new_vm_ref` オブジェクトの `is_a_template` フィールドが `false` に設定されます。これは、`new_vm_ref` が、標準的な (起動可能な) 仮想マシンであることを示しています。

#### 注:

このコールの実行中にテンプレートのディスクイメージが作成されるため、プロビジョニング操作に数分かかることがあります。この Debian テンプレートの例では、この段階で、新たに作成されたディスクに Debian ルートファイルシステムも実際に設定されます。

### 3.1.4. 仮想マシンの起動/一時停止/再開/停止のサイクル

新たにインストールされた仮想マシンのオブジェクトリファレンスを取得できたので、仮想マシンのライフサイクル操作も簡単に実行できます。

- 仮想マシンを起動するには、`VM.start(session, new_vm_ref)` をコールします。
- 実行中の仮想マシンを一時停止するには、`VM.suspend(session, new_vm_ref)` をコールします。
- 一時停止した仮想マシンを再開するには、`VM.resume(session, new_vm_ref)` をコールします。
- 仮想マシンを完全にシャットダウンするには、`VM.shutdown(session, new_vm_ref)` をコールします。

### 3.1.5. ログアウト

アプリケーションでの XenServer ホスト操作が終了したら、**Session.logout(session)**をコールすることをお勧めします。このコールにより、セッションリファレンスが無効になり、後続の API コールで使用できないようになります。また、セッションオブジェクトを格納していたサーバー側のメモリがこのコールにより解放されます。

非アクティブなセッションは最終的にタイムアウトになりますが、各 username または originator のサーバーの同時セッション数は、ハードコードにより 500 までに制限されています。この制限に達した後で新たなログインが発生すると、最も古いセッションオブジェクトが削除され、オブジェクトに関連付けられていたセッションリファレンスが無効になります。ほかのアプリケーションとの互換性を維持するために、サーバーへの同時アクセスに関して、以下の操作を実行することをお勧めします。

- アプリケーションとそのバージョンを識別する文字列を選択します。
- その識別文字列を **Session.login\_with\_password** の originator パラメーターに使用して、最初に単一のセッションを作成します。
- アプリケーションを実行している間はこのセッションを使用して（単一セッションを複数の異なるクライアントサーバー間ネットワーク接続で使用できます）、操作が終了したら明示的にログアウトします。

記述が不適切なクライアントでセッションがリークするか、または制限を超えた場合は、クライアントが適切な originator パラメーターを使用している限り、XenServer ログから簡単に識別することができ、正常に動作していないクライアントで最も長い時間アイドル状態であったセッションのみが破棄されます。この場合、そのクライアントで問題が発生することがありますが、他のクライアントは正常に動作します。正常に動作していないクライアントで originator を指定しなかった場合は、識別するのが困難になり、originator を指定しなかったすべてのクライアントのセッションも早い段階で破棄されます。

### 3.1.6. インストールと起動の例：概要

ここまで、API を使用して、XenServer テンプレートから仮想マシンをインストールし、その上で多数のライフサイクル操作を実行する方法について説明しました。これらの操作を実行するために必要なコール数が少ないことに注目してください。

- セッションを取得するためのコール：**Session.login\_with\_password()**。
- XenServer 上の仮想マシン（およびテンプレート）オブジェクトをクエリするためのコール：**VM.get\_all\_records()**。このコールで返された情報を使用して、インストールするテンプレートを選択しました。
- 選択したテンプレートから仮想マシンをインストールするための 2 つのコール：**VM.clone()**、続いて **VM.provision()**。
- 作成された仮想マシンを起動するコール：**VM.start()**（同様に、仮想マシンを一時停止、再開、およびシャットダウンのための各コール）。
- ログアウトのためのコール：**Session.logout()**。

以上の説明で重要なのは、API 自体は複雑で包括的な機能を備えていますが、一般的なタスク（仮想マシンの作成やライフサイクル操作など）の実行は非常に単純で、少数の簡単な API コールしか必要としない点です。次の節は最初は少々手ごわいかもしれませんが、この点を念頭に置いて学習を進めてください。

## 3.2. オブジェクトモデルの概要

ここでは、API のオブジェクトモデルの概要について説明します。ここで概説する各クラスのパラメーターとメソッドについては、『Citrix XenServer Management API』を参照してください。

最初に、API を構成するいくつかのコアクラスについて概要を説明します（最初に見たときに、これらの定義が抽象的に思えたとしても心配しないでください。後続の節の説明を読み、次の章でコード

サンプルを手順ごとに確認していくことで、これらの概念を具体的に理解できるようになるはずで  
す)。

VM	VM (仮想マシン) オブジェクトは、XenServer ホストまたはリソースプ ール上の特定の仮想マシンインスタンスを表します。メソッドには、 <b>start</b> 、 <b>suspend</b> 、および <b>pool_migrate</b> などがあります。パラメータ には、 <i>power_state</i> 、 <i>memory_static_max</i> 、および <i>name_label</i> などがあります (前節の例で、VM クラスで標準的な仮想マシンとテンプレートの両方を操 作できることを学習しました)。
Host	Host (ホスト) オブジェクトは、XenServer プール内の物理ホストを表 します。メソッドには、 <b>reboot</b> 、および <b>shutdown</b> などがあります。パ ラメータには、 <i>software_version</i> 、 <i>hostname</i> 、および (IP) <i>address</i> などがあ ります。
VDI	VDI オブジェクトは、仮想ディスクイメージを表します。仮想マシンに仮 想ディスクイメージを接続すると、仮想マシン内にブロックデバイスが現 れます。これにより、仮想ディスクイメージによりカプセル化されたビット を読み書きできるようになります。VDI クラスのメソッドには、 <i>resize</i> 、 <i>clone</i> などがあります。フィールドには、 <i>virtual_size</i> 、 <i>sharable</i> などがあります (前節の例で、仮想マシンテンプレート上で <b>VM.provision</b> をコールしたときに、新たに作成されたディスクを表すい くつかの VDI オブジェクトが自動的に作成され、VM オブジェクトに接続 されています)。
SR	SR (ストレージリポジトリ) オブジェクトは、仮想ディスクイメージの コレクションを集約し、仮想ディスクイメージのビットを格納している物 理ストレージのプロパティをカプセル化します。パラメータには、 <i>type</i> (XenServer でストレージリポジトリの仮想ディスクイメージの読み取 り/書き込みに使用するストレージ固有のドライバを指定します)、 <i>physical_utilisation</i> などがあります。メソッドには、 <b>scan</b> (ストレージ固 有のドライバを呼び出して、ストレージリポジトリに含まれる仮想ディス クイメージの一覧と、これらの仮想ディスクイメージのプロパティを取得 します)、 <b>create</b> (仮想ディスクイメージを格納できるように、物理スト レージのブロックを初期化します) などがあります。
Network	Network (ネットワーク) オブジェクトは、XenServer ホストのインス タンスが動作する環境に存在するレイヤ 2 ネットワークを表します。 XenServer ではネットワークを直接管理しないため、これは単に物理ネッ トワークトポロジと仮想ネットワークトポロジをモデル化するだけの軽 量なクラスです。特定の Network オブジェクトに VIF および PIF (下記 参照) で接続された VM オブジェクトおよび Host オブジェクトは、互い にネットワークパケットを送信できます。

これらのクラスについて完全に理解している場合は、次の章のコードの説明に進んでも構いません。  
既に説明したクラスの一部のみを使用して、多数の役立つアプリケーションを作成することができま  
す。理論的にクラスの説明を理解したい場合は、このまま読み進めてください。

上記のクラスに加えて、コネクタとして動作するさらに 4 つのクラスがあります。これらのクラスは、  
VM オブジェクト、Host オブジェクト、SR オブジェクト、および Network オブジェクトの間に関係  
を指定します。これらのクラスの最初の 2 つが *VBD* (仮想ブロックデバイス) と *VIF* (仮想ネットワ  
ークインターフェイス) で、それぞれ VM オブジェクトと VDI オブジェクト、および VM オブジェク  
トと Network オブジェクトの接続形式を決定します。

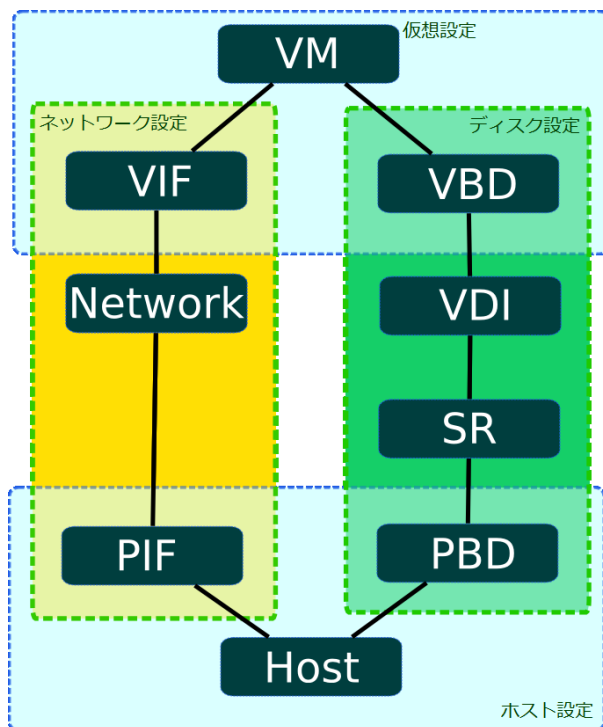


VBD	VBD (仮想ブロックデバイス) オブジェクトは、仮想マシンと仮想ディスクイメージの間の接続を表します。仮想マシンが起動すると、その VBD オブジェクトがクエリされ、接続するディスクイメージ (仮想ディスクイメージ) が決定されます。VBD クラスのメソッドには、plug (実行中の仮想マシンにディスクデバイスをホットプラグし、ディスクデバイスに含まれる指定された仮想ディスクイメージにアクセスできるようにします)、unplug (実行中の仮想マシンからディスクデバイスをホットアンプラグします) があります。フィールドには、device (指定された仮想ディスクイメージにアクセスできる、ゲスト内のデバイス名を決定します) があります。
VIF	VIF (仮想ネットワークインターフェイス) オブジェクトは、VM オブジェクトと Network オブジェクトの間の接続を表します。仮想マシンが起動すると、その VIF オブジェクトがクエリされ、作成するネットワークデバイスが決定されます。VIF クラスのメソッドには、plug (実行中の仮想マシンにネットワークデバイスをホットプラグします)、unplug (実行中の仮想マシンからネットワークデバイスをホットアンプラグします) があります。

「コネクタクラス」の残りの 2 つは、Host オブジェクトと Network オブジェクト、および Host オブジェクトと SR オブジェクトの接続形式を決定します。

PIF	PIF (物理ネットワークインターフェイス) オブジェクトは、Host オブジェクトと Network オブジェクトの間の接続を表します。ホストが (PIF を介して) Network オブジェクトに接続している場合、指定されたホストからのパケットを送受信できるようになります。PIF クラスのフィールドには、device (PIF オブジェクトに対応するデバイス名。eth0 など)、および MAC (PIF オブジェクトが表している NIC の MAC アドレス) があります。PIF オブジェクトは、物理ネットワークインターフェイスと VLAN の両方を表す点に注意してください (VLAN フィールドの値が正の整数であるインターフェイスが VLAN です)。
PBD	PBD (物理ブロックデバイス) オブジェクトは、Host オブジェクトと SR オブジェクトの間の接続を表します。フィールドには、currently-attached (指定された SR オブジェクトのストレージをホストで使用できるかどうか)、および device_config (指定されたホスト上で低レベルストレージデバイスを設定する方法を決定する、ストレージドライバー固有のパラメーターを指定します。たとえば、NFS ファイラで提供されるストレージリポジトリの場合、ファイラのホスト名と、ストレージリポジトリファイルを格納するファイラ上のパスを指定できます) があります。

図 3.1. 代表的な API クラス



仮想マシン、ホスト、ストレージ、およびネットワークを管理するための API クラスの概要

図 3.1. 「代表的な API クラス」は、仮想マシン、ホスト、ストレージ、およびネットワークの管理に関する API クラスの概要を示しています。この図から、ストレージとネットワークの設定そして仮想マシンとホストの設定に、対称性があることがわかります。

### 3.3. 仮想ネットワークインターフェイスおよび仮想ブロックデバイスの操作

ここでは、より複雑なシナリオを例にして、仮想ストレージとネットワークデバイスに関連するさまざまなタスクを API を使って実行する方法について説明します。

#### 3.3.1. ディスクの作成と仮想マシンへの接続

まず、新しい空のディスクイメージを作成し、実行中の仮想マシンにそのディスクイメージを接続します。既に実行中の仮想マシンがあり、その API オブジェクトリファレンスがわかっている（たとえば、前出の節で説明した手順で仮想マシンを作成し、サーバーがそのリファレンスを返している）とします。また、XenServer で認証済みで、セッションリファレンスを取得済みであるとしします。説明を簡潔にするために、この章の残りの部分ではセッションに関する説明を省略します。

##### 3.3.1.1. 新しい空のディスクイメージの作成

まず初めに、物理ストレージ上でディスクイメージのインスタンスを作成します。これを行うには、**VDI.create()** をコールします。**VDI.create** コールでは、多くのパラメータを指定できます。たとえば、以下のパラメータがあります。

- **name\_label** および **name\_description** : 人間が判読できる、ディスクの名前と説明です（ユーザーインターフェイスでの表示用など）。これらのフィールドは、特に必要がなければ空白のまま残してもかまいません。
- **SR** : 仮想ディスクイメージのビットを格納する物理ストレージを表す、ストレージリポジトリのオブジェクトリファレンスです。

- **read\_only** : このフィールドを true に設定すると、仮想ディスクイメージは読み取り専用形式でのみ仮想マシンに接続できます (**read\_only** フィールドが true の仮想ディスクイメージを読み取り/書き込み形式で接続しようとすると、エラーが返されます)。

**VDI.create** コールを呼び出すと、XenServer で物理ストレージ上に空のディスクイメージが作成されます。さらに、関連付けられた VDI オブジェクト (物理ストレージ上のディスクイメージを参照するデータモデルインスタンス) が作成され、この VDI オブジェクトへのリファレンスが返されます。

物理ストレージ上のディスクイメージを表す方法は、作成した仮想ディスクイメージの格納先ストレージリポジトリの種類によって異なります。たとえば、ストレージリポジトリの種類が lvm の場合、新しいディスクイメージは LVM ボリュームとして表されます。ストレージリポジトリの種類が nfs の場合、新しいディスクイメージは NFS ファイラ上に作成されるスパースな VHD ファイルとして表されます (API で **SR.get\_type()** コールを使用すると、ストレージリポジトリの種類をクエリできます)。

**注 :**

ストレージリポジトリの種類によっては、設定したブロックサイズで分割できるように *virtual-size* の値が切り上げられる可能性があります。

### 3.3.1.2. ディスクイメージの仮想マシンへの接続

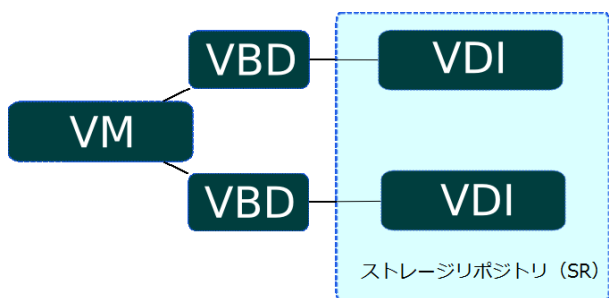
この時点で、実行中の仮想マシンと、作成したばかりの新しい仮想ディスクイメージが存在しています。これらはどちらも XenServer ホスト上に存在する独立したオブジェクトで、まだ関連付けられていません。そこで、次にこれらの仮想ディスクイメージと仮想マシンを関連付けるリンクを作成します。

接続は、VBD (仮想ブロックデバイス) オブジェクトと呼ばれる新しい「コネクタ」オブジェクトを作成することで確立されます。VBD オブジェクトを作成するには、**VBD.create()** コールを呼び出します。**VBD.create()** コールでは、多くのパラメータを指定できます。たとえば、以下のパラメータがあります。

- **VM** : 仮想ディスクイメージを接続する仮想マシンのオブジェクトリファレンスです。
- **VDI** : 接続する仮想ディスクイメージのオブジェクトリファレンスです。
- **mode** : 仮想ディスクイメージを読み取り専用形式または読み取り/書き込み形式のどちらで接続するかを指定します。
- **userdevice** : ゲスト内のブロックデバイスを指定します。仮想マシン内で実行するアプリケーションは、ブロックデバイスを通じて仮想ディスクイメージのビットを読み書きできます。
- **type** : 仮想ディスクイメージを仮想マシン内で標準的なディスクとして表示するか、CD として表示するかを指定します (このフィールドは、Linux 仮想マシンより Windows 仮想マシンでより重要な意味を持ちますが、この章では詳細について割愛します)。

**VBD.create** を呼び出すと、XenServer 上に VBD オブジェクトが作成され、そのオブジェクトリファレンスが返されます。ただし、このコール自体が実行中の仮想マシンに副次的な影響を与えることはありません (実行中の仮想マシン内にブロックデバイスは作成されません)。VBD オブジェクトが存在していてもゲスト内のブロックデバイスがアクティブでないということは、VBD オブジェクトの **currently\_attached** フィールドが false に設定されているということです。

図 3.2. 2 つの VDI オブジェクトが関連付けられた VM オブジェクト



2 つの VDI オブジェクトが関連付けられた VM オブジェクト

図 3.2. 「2 つの VDI オブジェクトが関連付けられた VM オブジェクト」は、VM、VBD、VDI、および SR の各オブジェクトの間関係を示しています。この例では、VM オブジェクトに 2 つの VDI オブジェクトが接続されています。また、VM オブジェクトとその VDI オブジェクトの間の接続を形成する 2 つの VBD オブジェクトがあり、VDI オブジェクトは同じ SR オブジェクト内に存在しています。

### 3.3.1.3. 仮想ブロックデバイスのホットプラグ

この段階で仮想マシンを再起動すると、VBD オブジェクトに対応するブロックデバイスが現れます。XenServer は、仮想マシンの起動時にすべての仮想ブロックデバイスをクエリし、対応する各仮想ディスクイメージをアクティブに接続します。

仮想マシンの再起動は非常に良い手段ですが、ここでは実行中の仮想マシンに新しい空のディスクを接続しようとしていることを思い出してください。これは、新しい VBD オブジェクトの **plug** メソッドを呼び出すことで達成できます。**plug** コールが正常終了すると、VBD オブジェクトが関連付けられているブロックデバイスが実行中の仮想マシン内部に現れます。これは実行中の仮想マシンから見ると、新しいディスクデバイスがホットプラグされたゲストオペレーティングシステムに認識させることとなります。このことを API 側から見ると、VBD オブジェクトの **currently\_attached** フィールドが **true** に設定されるということになります。

当然のことながら、VBD オブジェクトの **plug** メソッドには、対になる **unplug** メソッドがあります。VBD オブジェクトの **unplug** メソッドを呼び出すと、実行中の仮想マシンからブロックデバイスがホットアンプラグされ、それに応じて、VBD オブジェクトの **currently\_attached** フィールドが **false** に設定されます。

### 3.3.2. ネットワークデバイスの作成と仮想マシンへの接続

仮想マシンの仮想ネットワークインターフェイスの設定に関連する API コールは、多くの面で仮想ディスクデバイスの設定に関連するコールに似ています。そのため、ここでは、API オブジェクトモデルを使用してネットワークインターフェイスを作成する方法について詳細に説明することはせず、代わりに、仮想ネットワークデバイスの設定と仮想ストレージデバイスの設定の対称性について簡単に概説します。

ネットワークにおいて、VBD クラスに相当するのが VIF クラスです。API において、VBD オブジェクトが仮想マシン内のブロックデバイスを表現するものであるのと同様に、VIF オブジェクト（仮想ネットワークインターフェイス）は仮想マシン内のネットワークデバイスを表現するものです。VBD オブジェクトが VM オブジェクトと VDI オブジェクトを関連付けるのに対し、VIF オブジェクトは VM オブジェクトと Network オブジェクトを関連付けます。VBD オブジェクトと同様に、VIF オブジェクトにも、VIF オブジェクトに関連付けられている（ゲスト内の）ネットワークデバイスが現在アクティブなのか非アクティブなのかを決定する、**currently\_attached** フィールドがあります。仮想マシンの起動時にその仮想マシンの VIF オブジェクトがクエリされ、各 VIF オブジェクトに対応するネットワークデバイスが仮想マシン内で作成される点も VBD オブジェクトに似ています。同様に、VIF オブジェクトにも、実行中の仮想マシンでネットワークデバイスをホットプラグまたはホットアンプラグするための **plug** メソッドと **unplug** メソッドがあります。



### 3.3.3. ネットワークとストレージのためのホストの設定

ここまでで、仮想マシン内でブロックデバイスとネットワークデバイスの設定を管理するために、それぞれ VBD クラスと VIF クラスを使用することについて学習しました。ホストのストレージとネットワークを管理するには、これらに類似する 2 つのクラスを使用します。PBD（物理ブロックデバイス）と PIF（物理ネットワークインターフェイス）です。

#### 3.3.3.1. ホストのストレージ設定 : PBD

まず、PBD クラスから説明します。**PBD\_create()** コールでは、以下のようなパラメータを指定できます。

パラメーター	説明
host	物理ブロックデバイスを使用できる物理コンピュータ
SR	物理ブロックデバイスが接続するストレージリポジトリ
device_config	ホストのストレージリポジトリ用バックエンドドライバに提供される文字列マップで、物理ストレージデバイスの設定に必要な低レベルパラメータを含みます。このデバイス上でストレージリポジトリが認識されます。 <i>device_config</i> フィールドの内容は、物理ブロックデバイスが接続するストレージリポジトリの種類によって異なります ( <b>xe sm-list</b> コマンドを実行すると、ストレージリポジトリの種類の一覧を確認できます。この一覧の <i>configuration</i> フィールドによって、各種のストレージリポジトリで指定できる <i>device_config</i> パラメータの値が決定されます)。

たとえば、種類が `nfs`（仮想ディスクイメージが VHD ファイルとして格納される NFS ファイラ上のディレクトリ）の SR オブジェクト `s` があり、Host オブジェクト `h` から `s` にアクセスできるようにするとします。この場合、**PBD.create()** を呼び出して、ホスト `h`、ストレージリポジトリ `s`、および *device\_config* パラメータの値（以下のマップ）を指定します。

```
("server", "my_nfs_server.example.com"), ("serverpath", "/scratch/mysrs/sr1")
```

このパラメーターは、ストレージリポジトリ `s` がホスト `h` 上でアクセス可能であり、さらに、ストレージリポジトリ `s` にアクセスするために、NFS サーバー `my_nfs_server.xensource.com` 上のディレクトリ `/scratch/mysrs/sr1` をマウントする必要があることを XenServer ホストに通知します。

VBD オブジェクトと同様に、PBD オブジェクトにもアクティブなのか非アクティブなのかを決定する **currently\_attached** フィールドがあります。**PBD.plug** メソッドおよび **PBD.unplug** メソッドを呼び出すことで、ストレージリポジトリを特定のホストに接続したり切断したりすることができます。

#### 3.3.3.2. ホストのネットワーク設定 : PIF

ホストのネットワーク設定には、PIF オブジェクトを使用します。PIF オブジェクトが Network オブジェクト `n` を Host オブジェクト `h` に接続する場合、`n` に対応するネットワークは、PIF オブジェクトのフィールドで指定される物理ネットワークインターフェイス（または、VLAN タグを加えた物理ネットワークインターフェイス）にブリッジされます。

たとえば、Host オブジェクト `h` を Network オブジェクト `n` に接続する PIF オブジェクトが存在し、PIF オブジェクトの **device** フィールドが `eth0` に設定されているとします。これは、ネットワーク `n` 上のすべてのパケットが、ホストのネットワークデバイス `eth0` の物理 NIC にブリッジされることを意味します。

## 3.4. 仮想マシンのエクスポートとインポート

仮想マシンをファイルにエクスポートし、それを XenServer ホストにインポートすることができます。エクスポートプロトコルはシンプルな HTTP (S) GET であり、仮想マシンがプールのメンバ上

にある場合はマスタ上で実行する必要があります。標準 HTTP 基本認証で認証するか、セッションリファレンスを取得済みの場合はそのセッションを使用できます。エクスポートする仮想マシンを指定するには、UUID またはリファレンスを使用します。エクスポートを追跡するには、タスクを作成し、そのリファレンスを使用できます。仮想マシンのディスクにブールメンバからのみアクセスできる場合は、要求がリダイレクトされる可能性があります。

次の引数がコマンドラインで渡されます。

引数	説明
session_id	認証に使用するセッションのリファレンス。HTTP 基本認証を使用しない場合にのみ必要です。
task_id	操作を追跡するタスクオブジェクトのリファレンス。オプションであり、エクスポートを追跡するタスクオブジェクトを作成した場合にのみ必要です。
ref	仮想マシンのリファレンス。UUID を使用しない場合にのみ必要です。
uuid	仮想マシンの UUID。リファレンスを使用しない場合にのみ必要です。

次は、Linux コマンドラインツール cURL を使用する例です。

```
curl http://root:foo@myxenserver1/export?uuid=<vm_uuid> -o <exportfile>
```

このコマンドにより、指定した仮想マシンがファイル exportfile にエクスポートされます。

メタデータのみをエクスポートするには、URI として http://server/export\_metadata を使用します。

インポートプロトコルも同様で、HTTP (S) PUT が使用されます。session\_id および task\_id 引数はエクスポートの場合と同じです。ref および uuid は使用しません。仮想マシンのリファレンスと UUID は新たに生成されます。次の追加パラメータがあります。

引数	説明
restore	true の場合、元の仮想マシンの置き換え処理としてインポートが扱われます。VIF オブジェクト上の MAC アドレスがエクスポート時と変わらないため、元の仮想マシンが実行中の場合は競合が発生します。
force	true の場合、チェックサムエラーが無視されます（デフォルトでは、チェックサムエラーが検出された場合に仮想マシンが破棄されます）。
sr_id	仮想マシンのインポート先の SR オブジェクトのリファレンス。デフォルトでは、Pool.default_SR にインポートされます。

もう一度、cURL を使用する例を次に示します。

```
curl -T <exportfile> http://root:foo@myxenserver2/import
```

このコマンドにより、サーバー上のデフォルトのストレージリポジトリに仮想マシンがインポートされます。

**注：**

sr\_uuid を指定せず、かつデフォルトのストレージリポジトリが未設定の場合、「DEFAULT\_SR\_NOT\_FOUND」というエラーメッセージが返されます。

もう 1 つ例を示します。

```
curl -T <exportfile> http://root:foo@myxenserver2/import?sr_id=<opaque_ref_of_sr>
```

このコマンドにより、サーバー上の指定されたストレージリポジトリに仮想マシンがインポートされます。

メタデータのみをインポートするには、URI として `http://server/import_metadata` を使用します。

### 3.4.1. Xen 仮想アプライアンス (XVA) 仮想マシンのインポート形式

XenServer では、人間が判読できる従来の仮想マシン入力形式である XVA がサポートされます。ここでは、XVA の構文と構造について説明します。

XVA は、XML メタデータとディスクイメージのセットを含んでいる単一のディレクトリで構成されます。XVA 形式の仮想マシンは、直接実行するためのものではありません。XVA パッケージ内のデータは圧縮されており、永続ストレージにアーカイブするか、パッケージを展開して実行できる仮想マシンサーバー (XenServer ホストなど) に転送することを目的としています。

XVA はハイパーバイザーの種類に依存しないパッケージ形式なので、ほかのプラットフォームで XVA 仮想マシンのインスタンスを作成する、簡単なツールを作成できます。XVA では特定の実行時形式を指定しません。たとえば、ファイルイメージ、LVM ボリューム、QCoW イメージ、VMDK イメージ、または VHD イメージとしてディスクのインスタンスを作成できます。XVA 仮想マシンは何度でもインスタンス化でき、そのたびに異なる実行時形式にすることができます。

XVA では次のことを行いません。

- シリアル化またはトランスポートの特定の形式を指定する。
- 仮想マシン (またはテンプレート) のインストール時のカスタマイズ用メカニズムを提供する。
- インストール後に仮想マシンをアップグレードする方法に対応する。
- アプライアンスとして動作する複数の仮想マシンが通信する方法を定義する。

これらの問題は、関連する Open Virtual Appliance の仕様で対処されます。

XVA は単一のディレクトリであり、このディレクトリには必ず `ova.xml` と呼ばれるファイルが含まれています。このファイルには、XVA に含まれる仮想マシンが記述されます。詳しくは、第 3.2 節を参照してください。ディスクはサブディレクトリに格納され、`ova.xml` から参照されます。ディスクデータの形式については、第 3.3 節を参照してください。

この章の残りのページでは、次の用語を使用します。

- HVM : ハードウェアの仮想化サポートを活用して、オペレーティングシステムのカーネルを変更しないで実行するモード。
- PV : ハードウェアの仮想化サポートが不要な、特別に変更された「準仮想化」カーネルをハイパーバイザー上で明示的に実行するモード。

`ova.xml` ファイルには次の要素が含まれます。

```
<appliance version="0.1">
```

`version` 属性の値は、XVA の仕様のバージョンを示します。この場合は 0.1 です。`<appliance>` 要素には `<vm>` 要素が 1 つだけ存在します (Open Virtual Appliance の仕様では、複数の `<vm>` 要素を指定できます)。

```
<vm name="name">
```

各 `<vm>` 要素により単一の仮想マシンが定義されます。`name` 属性は将来的な内部使用を目的としており、`ova.xml` ファイル内で固有である必要があります。`name` 属性の値として、任意の有効な UTF-8 文字列を設定できます。各 `<vm>` 要素には次の必須要素があります。

```
<label>... text ... </label>
```



ユーザーインターフェイスに表示される仮想マシンの短い名前です。

```
<shortdesc> ... description ... </shortdesc>
```

ユーザーインターフェイスに表示される仮想マシンの説明です。<label>要素および<shortdesc>要素では、文字列の先頭と末尾の空白が無視されます。

```
<config mem_set="268435456" vcpus="1"/>
```

<config>要素には、仮想マシンのメモリサイズをバイト単位で表す mem\_set 属性と、CPU 数を表す vcpus 属性があります。

各<vm>要素にはブロックデバイスを表す<vbd>要素が含まれます。この要素の有無や数は任意で、次の形式で記述されます。

```
<vbd device="sda" function="root" mode="w" vdi="vdi_sda"/>
```

次の属性があります。

device	仮想マシンに認識される物理デバイスの名前です。Linux ゲストには sd[a-z]、Windows ゲストには hd[a-d]を使用します。
function	値が root の場合は、このディスクを使用してゲストを起動します (Linux のルート、つまり [/] のファイルシステムが存在するという意味ではないことに注意してください)。root が設定されたデバイスは 1 つのみである必要があります。仮想マシンの起動については、第 3.4 節を参照してください。ほかの文字列は無視されます。
mode	デバイスが読み取り/書き込み可能である場合の値は w、読み取り専用の場合の値は ro です。
vdi	ブロックデバイスが接続するディスクイメージの名前です (<vdi>要素で表わされます)。

各<vm>要素に、<hacks is\_hvm="false" kernel\_boot\_cmdline="root=/dev/sda1 ro"/>のような<hacks>要素が含まれる場合があります。この<hacks>要素は XenServer で生成される XVA ファイルに存在しますが、将来削除される予定です。is\_hvm 属性には、仮想マシンを HVM モードで起動する必要があるかどうかを示す true または false が指定されます。kernel\_boot\_cmdline 属性には、pygrub でゲストを起動するときの追加のカーネルコマンドライン引数が含まれます。

<appliance>要素には、<vm>要素のほか、<vdi>要素が含まれます。この要素の有無や数は任意で、次の形式で記述されます。

```
<vdi name="vdi_sda" size="5368709120" source="file://sda" type="dir-gzipped-chunks">
```

各<vdi>要素により、単一のディスクイメージが定義されます。次の属性があります。

- name : 仮想ディスクイメージの名前で、<vbd>要素の vdi 属性により参照されます。任意の有効な UTF-8 文字列を設定できます。
- size : 必要なイメージのバイト単位のサイズです。
- source : イメージデータの場所を記述する URI です。現在設定できるのは「file://」で始まる URI のみで、ova.xml を格納するディレクトリからの相対パスである必要があります。
- ディスクデータの形式です (第 3.3 節を参照してください)。

type 属性の値が「dir-gzipped-chunks」の<vdi>要素により、単一ディスクイメージエンコーディングが定義されます。各イメージは、次のような一連のファイルを含むディレクトリによって表わされます。

```
-rw-r--r-- 1 dscott xendev 458286013 Sep 18 09:51 chunk000000000.gz
-rw-r--r-- 1 dscott xendev 422271283 Sep 18 09:52 chunk000000001.gz
-rw-r--r-- 1 dscott xendev 395914244 Sep 18 09:53 chunk000000002.gz
-rw-r--r-- 1 dscott xendev 9452401 Sep 18 09:53 chunk000000003.gz
-rw-r--r-- 1 dscott xendev 1096066 Sep 18 09:53 chunk000000004.gz
-rw-r--r-- 1 dscott xendev 971976 Sep 18 09:53 chunk000000005.gz
-rw-r--r-- 1 dscott xendev 971976 Sep 18 09:53 chunk000000006.gz
-rw-r--r-- 1 dscott xendev 971976 Sep 18 09:53 chunk000000007.gz
-rw-r--r-- 1 dscott xendev 573930 Sep 18 09:53 chunk000000008.gz
```

chunk-XXXXXXXXX.gz という名前の各ファイルは、1e9 バイト (1GB、1GiB ではない) のローブロックデータを含んだ GZIP ファイルです。一部のファイルシステムでのファイルサイズの上限を超えないように、この小さなサイズが選択されています。ファイルを展開して結合すると、元のイメージが復元されます。

XenServer では 2 つのメカニズムで仮想マシンを起動できます。1 つは pygrub で抽出された準仮想化カーネルを使用する方法で、もう 1 つは HVM モードを使用する方法です。現在の実装では、<hacks>要素の is\_hvm 属性で指定されたメカニズムが使用されます。

この節の残りのページでは、XVA としてパッケージ化されたシンプルな Debian 仮想マシンを例にして説明します。この仮想マシンには 2 つのディスクがあります。1 つは 5,120MiB のルートファイルシステムで、pygrub でゲストを起動するために使用されます。もう 1 つは 512MiB で、スワップに使用されます。メモリは 512MiB で、仮想 CPU を 1 つ使用します。

この Debian 仮想マシンの最上位ディレクトリは、次のように表示されます。

```
$ ls -l
total 4
drwxr-xr-x 3 dscott xendev 4096 Oct 24 09:42 very simple Debian VM
```

この最上位ディレクトリ内には、次のように、各ディスク用の 2 つのサブディレクトリと、単一の ova.xml ファイルがあります。

```
$ ls -l very\ simple\ Debian\ VM/
total 8
-rw-r--r-- 1 dscott xendev 1016 Oct 24 09:42 ova.xml
drwxr-xr-x 2 dscott xendev 4096 Oct 24 09:42 sda
drwxr-xr-x 2 dscott xendev 4096 Oct 24 09:53 sdb
```

各ディスクのサブディレクトリには、GZIP で圧縮された 1GB のローディスクブロックであるいくつかのファイルがあります。

```
$ ls -l very\ simple\ Debian\ VM\sda/
total 2053480
-rw-r--r-- 1 dscott xendev 202121645 Oct 24 09:43 chunk-000000000.gz
-rw-r--r-- 1 dscott xendev 332739042 Oct 24 09:45 chunk-000000001.gz
-rw-r--r-- 1 dscott xendev 401299288 Oct 24 09:48 chunk-000000002.gz
-rw-r--r-- 1 dscott xendev 389585534 Oct 24 09:50 chunk-000000003.gz
-rw-r--r-- 1 dscott xendev 624567877 Oct 24 09:53 chunk-000000004.gz
-rw-r--r-- 1 dscott xendev 150351797 Oct 24 09:54 chunk-000000005.gz
```

```
$ ls -l very\ simple\ Debian\ VM\sdb
total 516
-rw-r--r-- 1 dscott xendev 521937 Oct 24 09:54 chunk-000000000.gz
```

この Debian 仮想マシンの ova.xml には、次のような内容が記述されます。





```
<?xml version="1.0" ?>
<appliance version="0.1">
  <vm name="vm">
    <label>
      very simple Debian VM
    </label>
    <shortdesc>
      the description field can contain any valid UTF-8
    </shortdesc>
    <config mem_set="536870912" vcpus="1"/>
    <hacks is_hvm="false" kernel_boot_cmdline="root=/dev/sda1 ro">
      <!--This section is temporary and will be ignored in future. Attribute
      is_hvm ("true" or "false") indicates whether the VM will be booted in HVM mode. In
      future this will be autodetected. Attribute kernel_boot_cmdline contains the kernel
      commandline for the case where a proper grub menu.lst is not present. In future
      booting shall only use pygrub.-->
    </hacks>
    <vbd device="sda" function="root" mode="w" vdi="vdi_sda"/>
    <vbd device="sdb" function="swap" mode="w" vdi="vdi_sdb"/>
  </vm>
  <vdi name="vdi_sda" size="5368709120" source="file://sda" type="dir-gzippedchunks"/>
  <vdi name="vdi_sdb" size="536870912" source="file://sdb" type="dir-gzippedchunks"/>
</appliance>
```

## 3.5. XML-RPC に関する注意事項

### 3.5.1. datetime

この API における datetime の制御方法は、XML-RPC 仕様に従いません。API では、datetime 文字列の後に「Z」が付加されます。これにより、時刻が UTC 形式であることを示します。

## 3.6. 参考資料

この章では、API とそのオブジェクトモデルについての概要を学習しました。この章の目的は、API の詳細を説明することではなく、次の章のコードサンプルを読解し、さらに『Citrix XenServer Management API』の詳細な情報を理解するための背景知識を提供することです。

より詳細な情報は、さまざまな場所から入手できます。

- 『XenServer 管理者ガイド』には、**xe** CLI の概要が記載されています。多くの **xe** コマンドでは API がそのまま使用されるため、**xe** を使用することは、この章で説明した API オブジェクトモデルを理解する手がかりになります。
- 次の章のコードサンプルでは、さまざまなクライアント言語での API コーディングの具体例について説明します。
- 『Citrix XenServer Management API』では、プログラミング上の API の動作と XML/RPC メッセージの形式についてより詳しく説明します。
- XenServer ホストの dom0 自体に、API を使用するスクリプトが含まれています。たとえば、`/opt/xen-source/libexec/shutdown` は、仮想マシンを完全にシャットダウンする Python プログラムです。このスクリプトは、ホスト自体をシャットダウンするときに呼び出されます。

## 第 4 章 API の使用

この章では、実際のプログラムで XenServer 管理 API を使用して XenServer ホストおよび仮想マシンを管理する方法について説明します。まず、典型的なクライアントアプリケーションについて順を追って説明し、API を使用して一般的なタスクを実行する方法を具体的に示します。サンプルコードでは Python の構文が用いられていますが、ほかのプログラミング言語でも同様の方法を使用できます。言語バインディング自体については後述します。章の最後では、2 つの完全な例について解説します。

### 4.1. 典型的なアプリケーションの構造

ここでは、XenServer 管理 API を使用する典型的なアプリケーションの構造について説明します。大半のクライアントアプリケーションは、XenServer ホストに接続し（ユーザー名とパスワードを使用するなどして）認証を受ける処理から始まります。認証に成功すると、サーバーで「セッション」オブジェクトが作成され、クライアントへリファレンスが返されます。このリファレンスは引数として、それ以降のすべての API コールに渡されます。いったん認証されると、クライアントでほかのオブジェクト（XenServer ホストや仮想マシンなど）へのリファレンスを検索して、それらに対する操作を呼び出すことができます。操作は、同期的または非同期的に呼び出すことができます。非同期操作の場合は、特殊なタスクオブジェクトが操作の状態と進行状況を表します。これらのアプリケーション要素については、次の節で詳しく説明します。

#### 4.1.1. 低レベルトランスポートの選択

API コールは、2 つのトランスポートで実行できます。

- IP ネットワークを介したポート 443 (HTTPS) 上の SSL 暗号化 TCP トランスポート
- ローカル Unix ドメインソケット `/var/xapi/xapi` を介したプレーンテキストトランスポート

SSL 暗号化 TCP トランスポートはホスト外部からのすべてのトラフィックで使用され、UNIX ドメインソケットは XenServer ホスト上で実行されているサービスで使用されます。SSL 暗号化 TCP トランスポートでは、すべての API コールをプールマスタに送信する必要があります。送信できないと、`HOST_IS_SLAVE` エラーが返されます。このエラーには、パラメータとしてマスタの IP アドレスが含まれます。

高可用性が有効なプールなどでは、プールマスタとして動作するホストが変更される場合があります。したがって、クライアントでは次のステップを実装して、プールマスタの変更を検出し、必要に応じて新しいマスタに接続するようにしてください。

#### プールマスタの変更に対処するには

1. ホストサーバー一覧の更新をサブスクライブし、プール内のホストの現在の一覧を保持します。
2. プールマスタへの接続に失敗した場合は、応答があるまで一覧内のすべてのホストへの接続を試行します。
3. 最初に応答するホストから `HOST_IS_SLAVE` エラーが返されます。このメッセージには、(そのホストが新しいマスタでない場合) 新しいプールマスタの識別情報が含まれます。
4. 新しいプールマスタに接続します。

#### 注：

例外として、UNIX ドメインソケットを介して送信されるすべてのメッセージは、正しいホストに透過的に転送されます。

### 4.1.2. 認証とセッションの処理

多くの API コールは、第 1 パラメータとしてセッションリファレンスを取ります。有効なリファレンスを指定しないと、`SESSION_INVALID` エラーが返されます。セッションリファレンスは、`login_with_password` 関数にユーザー名とパスワードを指定して取得します。

#### 注：

例外として、ローカル UNIX ドメインソケットを介してコールが実行された場合、ユーザー名とパスワードは無視され、常にコールは成功します。

各セッションにはタイムスタンプの `last active` フィールドがあり、これは API コールごとに更新されます。XenServer には、アクティブなセッション数の上限を 500 とする制限が組み込まれています。指定した `username` または `originator` でこの制限を超えると、`last active` フィールドの値が最も古いセッションが削除されます。さらに、`last active` フィールドの値が 24 時間より古いセッションもすべて削除されます。したがって、次の重要事項を守る必要があります。

- ログイン時には、適切な `originator` を指定する。
- セッションリークを防ぐために、必ずアクティブなセッションからログアウトする。
- `SESSION_INVALID` エラーが返された場合にサーバーに再度ログインするようにする。

次の Python サンプルコードでは、UNIX ドメインソケットを介した接続が確立され、セッションが作成されます。

```
import XenAPI

session = XenAPI.xapi_local()
try:
    session.xenapi.login_with_password("root", "", "2.3", "My Widget v0.1")
    ...
finally:
    session.xenapi.session.logout()
```

### 4.1.3. オブジェクトリファレンスの検索

アプリケーションが認証されたら、次に、オブジェクトの状態をクエリしたりオブジェクトに対する操作を呼び出したりするために、オブジェクトへのリファレンスを取得します。すべてのオブジェクトに、次のような「暗黙」のメッセージセットがあります。

- **get\_by\_name\_label** : 特定のラベルを持つ特定のクラスのすべてのオブジェクトの一覧を返します。
- **get\_by\_uuid** : UUID によって特定される単一のオブジェクトを返します。
- **get\_all** : 特定のクラスのすべてのオブジェクトへのリファレンスセットを返します。
- **get\_all\_records** : 特定のクラスの各オブジェクトのレコードへのリファレンスマップを返します。

たとえば、すべてのホストを一覧するには、次のように指定します。

```
hosts = session.xenapi.host.get_all()
```

「my first VM」という名前のすべての仮想マシンを検索するには、次のように指定します。

```
vms = session.xenapi.VM.get_by_name_label('my first VM')
```

#### 注：

オブジェクトの `name label` フィールドの値は固有であるとは限らないため、**get\_by\_name\_label** API コールは、単一のリファレンスではなくリファレンスセットを返します。



上記のオブジェクト検索メソッドに加え、ほとんどのオブジェクトでも、ほかのオブジェクトへのリファレンスがフィールドに含まれています。たとえば、次のコールによって、特定のホスト上で実行中の仮想マシンのセットを検索できます。

```
vms = session.xenapi.host.get_resident_VMs(host)
```

#### 4.1.4. オブジェクトに対する同期操作の呼び出し

オブジェクトリファレンスを取得したら、それらに対する操作を呼び出すことができます。たとえば、仮想マシンを開始するには、次のように指定します。

```
session.xenapi.VM.start(vm, False, False)
```

すべての API コールはデフォルトでは同期的に呼び出され、操作が完了または失敗するまで戻りません。たとえば、**VM.start** は、仮想マシンが起動を開始するまで戻りません。

##### 注：

**VM.start** コールが戻るときには、仮想マシンが起動を開始しています。いつ起動が終了したかを判別するには、ゲスト内エージェントが **VM\_guest\_metrics** オブジェクトで内部統計情報を報告するまで待機します。

#### 4.1.5. タスクを使用した非同期操作の管理

時間のかかる操作 (**VM.clone** や **VM.copy** など) の管理を簡素化するため、関数を同期 (デフォルト) と非同期の 2 通りの形式で使用できます。各非同期関数は、進行中の操作に関する情報を含む、以下のリファレンスを返します。

- 保留状態かどうか。
- 成功したか、失敗したか。
- 進行状況 (0 から 1 の範囲で表します)。
- 操作によって返される結果やエラーコード。

アプリケーションで **VM.clone** 操作の進行状況を追跡し、進行状況バーを表示するには、次のようなコードを使用します。

```
vm = session.xenapi.VM.get_by_name_label('my vm')
task = session.xenapi.Async.VM.clone(vm)
while session.xenapi.task.get_status(task) == "pending":
    progress = session.xenapi.task.get_progress(task)
    update_progress_bar(progress)
    time.sleep(1)
session.xenapi.task.destroy(task)
```

##### 注：

いわゆる「行儀のよい」クライアントアプリケーションを作成するには、結果またはエラーの読み取りが完了したら、非同期操作により作成されたタスクを必ず削除する必要があります。組み込みの上限値をタスク数が超えた場合、最も古い完了済みのタスクがサーバーにより削除されます。

#### 4.1.6. イベントのサブスクライブおよびリッスン

タスクおよび測定値クラスを除き、オブジェクトが変更されるたびに、サーバーでイベントが生成されます。クライアントは、頻繁にポーリングする代わりにクラス単位でこのイベントストリームをサブスクライブし、更新を受信できます。イベントには 3 つの種類があります。

- *add* : オブジェクトが作成されたときに生成されます。
- *del* : オブジェクトが破棄される直前に生成されます。

- `mod` : オブジェクトのフィールドが変更されたときに生成されます。

イベントには、単調増加 ID、オブジェクトクラスの名前、および `get_record()` の結果に相当するオブジェクト状態のスナップショットも含まれます。

クライアントは、クラス名一覧または特殊文字の「\*」を指定して `event.register()` をコールすることで、イベントを登録します。また、イベントが入手可能になるまで待機して新しいイベントを返す `event.next()` を実行することで、イベントを受信します。

#### 注 :

サーバー上の生成済みイベントのキューの長さは限定されているため、非常に低速なクライアントでは十分な速さでイベントを読み取れない場合があります。その場合は、EVENTS\_LOST エラーが返されます。クライアントには、この種の状況に対処する用意が必要です。そのためには、イベントを再登録する処理と、待機している条件が未登録の間に真になっていないことを確認する処理を追加します。

次のサンプル Python コードでは、システムによって生成された各イベントの概要が出力されます (同様のコードが `Xenserver-SDK/XenServerPython/samples/watch-all-events.py` に含まれています)。

```
fmt = "%8s %20s %5s %s"
session.xenapi.event.register(["*"])
while True:
    try:
        for event in session.xenapi.event.next():
            name = "(unknown)"
            if "snapshot" in event.keys():
                snapshot = event["snapshot"]
                if "name_label" in snapshot.keys():
                    name = snapshot["name_label"]
                print fmt % (event['id'], event['class'], event['operation'], name)
    except XenAPI.Failure, e:
        if e.details == ["EVENTS_LOST"] :
            print "Caught EVENTS_LOST; should reregister"
```

## 4.2. 言語バインディング

### 4.2.1. C

SDK の `XenServer-SDK/libxenserver/src` ディレクトリに、C 言語バインディングのソースと、バインディングをライブラリにコンパイルするメイクファイルが収録されています。各 API オブジェクトは、そのオブジェクトのすべての API 関数の宣言が含まれているヘッダーファイルに関連付けられています。たとえば、仮想マシンの操作を呼び出すために必要な型定義と関数は、すべて `xen_vm.h` に含まれています。

#### C バインディングの依存関係

サポートされるプラットフォーム :	Linux
ライブラリ :	言語バインディングは、C プログラムによってリンクされる <code>libxenserver.so</code> として生成されています。
依存関係 :	<ul style="list-style-type: none"> <li>• XML ライブラリ (GNU Linux 上の <code>libxml2.so</code>)</li> <li>• Curl ライブラリ (<code>libcurl2.so</code>)</li> </ul>

C バインディングの以下の単純なサンプルコードが収録されています。

- `test_vm_async_migrate` : 非同期 API コールを使用して、実行中の仮想マシンをメンバホストからプールマスタに移行します。

- test\_vm\_ops : ホスト機能をクエリして、仮想マシンを作成し、新しい空のディスクイメージを仮想マシンに接続した後、さまざまな電源操作を実行します。
- test\_failures : エラー文字列を enum\_xen\_api\_failure に変換したり、その逆を行ったりします。
- test\_event\_handling : 接続上のイベントをリスンします。
- test\_enumerate : さまざまな API オブジェクトを列挙します。

### 4.2.2. C#

C# バインディングは XenServer-SDK/XenServer.NET ディレクトリに収録されています。Microsoft Visual Studio でのビルドに適したプロジェクトファイルも提供されています。各 API オブジェクトは、1 つの C# ファイルに関連付けられています。たとえば仮想マシンの操作を実装する関数は、VM.cs ファイルに含まれています。

#### C# バインディングの依存関係

サポートされるプラットフォーム :	.NET Framework Version 2.0 をインストールした Windows
ライブラリ :	言語バインディングは、C# プログラムによってリンクされるダイナミックリンクライブラリである XenServer.dll として生成されています。
依存関係 :	XenServer.dll と XML-RPC サーバーの間の通信を可能にするために、CookComputing.XMLRpcV2.dll が必要です。Version 2.1.0.6 でテストされています。ほかのバージョンでも動作する場合がありますが、このバージョンを使用することをお勧めします。

C# バインディングの 3 つの単純なサンプルコードが、XenSdkSample.sln ソリューションの個別のプロジェクトとして XenServer-SDK/XenServer.NET/samples ディレクトリに収録されています。

- GetVariousRecords : XenServer ホストにログインし、ホスト、ストレージ、および仮想マシンに関する情報を表示します。
- GetVmRecords : XenServer ホストにログインし、すべての仮想マシンレコードのリストを出力します。
- VmPowerStates : XenServer ホストにログインし、仮想マシンを検索してさまざまな電源状態に切り替えます。シャットダウン状態の仮想マシンが必要です。

### 4.2.3. Java

Java バインディングは XenServer-SDK/XenServerJava ディレクトリに収録されています。Microsoft Visual Studio でのビルドに適したプロジェクトファイルも提供されています。各 API オブジェクトは、1 つの Java ファイルに関連付けられています。たとえば仮想マシンの操作を実装する関数は、VM.java ファイルに含まれています。

#### Java バインディングの依存関係

サポートされるプラットフォーム :	Linux および Windows
ライブラリ :	言語バインディングは、Java プログラムによってリンクされる Java アーカイブファイルである xenserver-6.5.jar として生成されています。
依存関係 :	<ul style="list-style-type: none"> <li>• xenserver.jar と XML-RPC サーバーの間の通信を可能にするために、xmlrpc-client-3.1.jar が必要です。</li> <li>• サンプルを実行するために、ws-commons-util-1.0.2.jar が必要です。</li> </ul>

メインの XenServer-SDK/XenServerJava/samples/RunTests.java ファイルを実行すると、同じディレクトリ内の以下の一連のサンプルが実行されます。

- AddNetwork : NIC に接続されていない新規内部ネットワークを追加します。
- SessionReuse : 1 つのセッションオブジェクトを複数の接続で共有します。
- AsyncVMCreate : 組み込みのテンプレートから新規仮想マシンを非同期的に作成して、その仮想マシンを起動して停止します。
- VdiAndSrOps : ストレージリポジトリや VDI に関するさまざまなテストを実行します (ダミーストレージリポジトリの作成など)。
- CreateVM : 1 つのネットワークと DVD ドライブを持つ仮想マシンをデフォルトストレージリポジトリ上に作成します。
- DeprecatedMethod : 廃止予定の API メソッドがコールされたときに警告が表示されるかどうかをテストします。
- GetAllRecordsOfAllTypes : すべての種類のオブジェクトのすべてのレコードを取得します。
- SharedStorage : 共有 NFS ストレージリポジトリを作成します。
- StartAllVMs : ホストに接続してそのホスト上の各仮想マシンを起動します。

#### 4.2.4. PowerShell

このリリースには、2 つのバージョンの PowerShell 言語バインディングが含まれています。

従来の PowerShell バインディングは、XenServer-SDK/XenPSSnapIn\_old ディレクトリに収録されています。Windows インストーラ XenServerPSSnapIn.msi と、XenServer API を Windows PowerShell 1.0 コマンドレットで使用するソースコードが提供されています。

##### 従来の PowerShell バインディングの依存関係

サポートされるプラットフォーム :	.NET Framework 3.5 と PowerShell 1.0 をインストールした Windows
ライブラリ :	XenServerPSSnapIn.dll
依存関係 :	XML-RPC サーバーとの通信を可能にするために、CookComputing.XMLRpcV2.dll が必要です。Version 2.1.0.6 でテストされています。ほかのバージョンでも動作する場合がありますが、このバージョンを使用することをお勧めします。

従来の PowerShell バインディングのサンプルスクリプトが、XenServer-SDK/XenPSSnapIn\_old/samples ディレクトリに収録されています。

- AutomatedTestCore.ps1 : XenServer ホストにログインして、ストレージリポジトリおよび仮想マシンを作成し、さまざまな電源操作を実行します。

新たに再設計された PowerShell バインディングは、XenServer-SDK/XenPSSnapIn ディレクトリに収録されています。Windows インストーラ XenServerPSSnapIn.msi と、XenServer API を Windows PowerShell 2.0 コマンドレットで使用するソースコードが提供されています。

##### PowerShell バインディングの依存関係

サポートされるプラットフォーム :	.NET Framework 3.5 と PowerShell 2.0 をインストールした Windows
ライブラリ :	XenServerPSSnapIn.dll

依存関係：	XML-RPC サーバーとの通信を可能にするために、CookComputing.XMLRpcV2.dll が必要です。Version 2.1.0.6 でテストされています。ほかのバージョンでも動作する場合がありますが、このバージョンを使用することをお勧めします。
-------	---

PowerShell バインディングのサンプルスクリプトが、XenServer-SDK/XenPSSnapIn/samples ディレクトリに収録されています。

- AutomatedTestCore.ps1：XenServer ホストにログインして、ストレージリポジトリおよび仮想マシンを作成し、さまざまな電源操作を実行します。

## 4.2.5. Python

Python バインディングは、単一のファイルの XenServer-SDK/XenServerPython/XenAPI.py に含まれています。

### Python バインディングの依存関係

サポートされるプラットフォーム：	Linux
ライブラリ：	XenAPI.py
依存関係：	なし

SDK には、Python の 8 つのサンプルコードが含まれています。

- fixpbds.py：共通ストレージにアクセスするための設定値を再設定します。
- install.py：Debian 仮想マシンをインストールし、それをネットワークに接続して起動し、仮想マシンにより IP アドレスが報告されるまで待機します。
- license.py：新しいライセンスを XenServer ホストにアップロードします。
- permute.py：仮想マシンのセットを選択し、XenMotion を使用してホスト間でそれらを同時に移行します。
- powercycle.py：仮想マシンのセットを選択し、それらの電源を入れ直します。
- shell.py：テスト用の単純なインタラクティブシェルです。
- vm\_start\_async.py：操作を非同期的に呼び出す方法を示します。
- watch-all-events.py：すべてのイベントを登録し、イベントが発生したときに詳細を出力します。

## 4.2.6. コマンドラインインターフェイス (CLI)

サードパーティのソフトウェア開発者は、未加工の XML-RPC や提供されている言語バインディングに加えて **xe** コマンドラインインターフェイスを使用して、ソフトウェアを XenServer に統合できます。xe CLI は、XenServer ホスト上にデフォルトでインストールされます。また、Linux ではスタンドアロンのリモート CLI も使用できます。Windows の場合は、xe.exe CLI 実行ファイルが XenCenter と一緒にインストールされます。

### CLI の依存関係

サポートされるプラットフォーム：	Linux および Windows
ライブラリ：	なし

バイナリ :	xe (Windows の場合は xe.exe)
依存関係 :	なし

CLI を使用すると、スクリプトやその他のプログラムからほとんどすべての API コールを直接呼び出し、必要なセッション管理を自動的に実行できます。xe CLI の構文と機能については、『[XenServer 管理者ガイド](#)』を参照してください。このほかの情報やサンプルコードについては、[Citrix Knowledge Center](#) を参照してください。

#### 注 :

XenServer ホストのコンソールから CLI を実行する場合は、コマンド名と引数の両方で Tab キーによる自動補完機能を使用できます。

## 4.3. 完全なアプリケーションの例

ここでは、API を使用する実際のプログラムの完全な例を 2 つ挙げて説明します。

### 4.3.1. XenMotion による複数の仮想マシンの同時移行

この Python のサンプル (XenServer-SDK/XenServerPython/samples/permute.py) は、XenMotion を使用して、複数の仮想マシンをリソースプール内のサーバーの間で同時に移行します。このサンプルでは、非同期 API コールを使用し、タスクセットが完了するまで待機します。

プログラムの冒頭には標準ボイラープレートテキストがあり、次に API バインディングモジュールをインポートします。

```
import sys, time
import XenAPI
```

次に、サーバーの URL、ユーザー名、パスワード、および反復処理の回数を含むコマンドライン引数を解析します。ユーザー名とパスワードはセッションを確立するときに使用されます。このセッションは main 関数に渡され、さらにループ内で複数回呼び出されます。try: と finally: が使用されていることに注目してください。これにより、プログラムの終了時に必ずセッションからログアウトします。

```
if __name__ == "__main__":
    if len(sys.argv) < 5:
        print "Usage:"
        print sys.argv[0], " <url> <username> <password> <iterations>"
        sys.exit(1)
    url = sys.argv[1]
    username = sys.argv[2]
    password = sys.argv[3]
    iterations = int(sys.argv[4])
    # First acquire a valid session by logging in:
    session = XenAPI.Session(url)
    session.xenapi.login_with_password(username, password, "2.3",
                                      "Example migration-demo v0.1")
    try:
        for i in range(iterations):
            main(session, i)
    finally:
        session.xenapi.session.logout()
```

main 関数は、システム内で実行中の各仮想マシンを調べ、コントロールドメイン (システムの一部でユーザーは制御できません) をフィルタ (除外) します。そして、実行中の仮想マシンとそれらの現在のホストの一覧を作成します。





```
def main(session, iteration):
    # Find a non-template VM object
    all = session.xenapi.VM.get_all()
    vms = []
    hosts = []
    for vm in all:
        record = session.xenapi.VM.get_record(vm)
        if not(record["is_a_template"]) and \
            not(record["is_control_domain"]) and \
            record["power_state"] == "Running":
            vms.append(vm)
            hosts.append(record["resident_on"])
    print "%d: Found %d suitable running VMs" % (iteration, len(vms))
```

次にホスト一覧をローテーションします。

```
# use a rotation as a permutation
hosts = [hosts[-1]] + hosts[:len(hosts)-1]
```

その後、このローテーションに従い、XenMotion を使って各仮想マシンを新しいホストに移行します（一覧の 2 番目のホストで実行中の仮想マシンを 1 番目のホストに移行するなど）。各移行処理を並行して実行するために、非同期版の **VM.pool\_migrate** を使用して、タスクリファレンスの一覧を作成します。**VM.pool\_migrate** に渡される *live* フラグに注意してください。これにより、仮想マシンを停止せずに移行することができます。

```
tasks = []
for i in range(0, len(vms)):
    vm = vms[i]
    host = hosts[i]
    task = session.xenapi.Async.VM.pool_migrate(vm, host, { "live": "true" })
    tasks.append(task)
```

その後、完了したことを確認するために、タスク一覧をポーリングします。

```
finished = False
records = {}
while not(finished):
    finished = True
    for task in tasks:
        record = session.xenapi.task.get_record(task)
        records[task] = record
        if record["status"] == "pending":
            finished = False
    time.sleep(1)
```

すべてのタスクが保留 (*pending*) 状態でなくなったら（正常終了、失敗、またはキャンセル状態になったら）、すべてのタスクが成功したかどうかを確認するために、もう一度ポーリングします。

```
allok = True
for task in tasks:
    record = records[task]
    if record["status"] <> "success":
        allok = False
```

タスクのいずれかが失敗した場合は詳細を出力し、例外を生成して、さらに調査するためにそのタスクオブジェクトを残します。すべてのタスクが成功した場合はタスクオブジェクトを破棄し、関数が戻ります。

```

if not(alloc):
    print "One of the tasks didn't succeed at", \
        time.strftime("%F:%HT%M:%SZ", time.gmtime())
    idx = 0
    for task in tasks:
        record = records[task]
        vm_name = session.xenapi.VM.get_name_label(vms[idx])
        host_name = session.xenapi.host.get_name_label(hosts[idx])
        print "%s : %12s %s -> %s [ status: %s; result = %s; error = %s ]" % \
            (record["uuid"], record["name_label"], vm_name, host_name, \
             record["status"], record["result"], repr(record["error_info"]))
        idx = idx + 1
    raise "Task failed"
else:
    for task in tasks:
        session.xenapi.task.destroy(task)

```

### 4.3.2. xe CLI による仮想マシンの複製

この **bash** スクリプトのサンプルは、xe CLI を使用して仮想マシンをシャットダウンしてから複製します。

プログラムの冒頭のボイラープレートでは、環境変数 *XE* が設定されているかどうかをチェックします。この環境変数が設定されている場合はその変数が CLI のフルパスを参照し、設定されていない場合は現在のパス上に xe CLI があるとして動作します。次に、サーバー名、ユーザー名、およびパスワードの入力をユーザーに促します。

```

# Allow the path to the 'xe' binary to be overridden by the XE environment variable
if [ -z "${XE}" ]; then
    XE=xe
fi

if [ ! -e "${HOME}/.xe" ]; then
    read -p "Server name: " SERVER
    read -p "Username: " USERNAME
    read -p "Password: " PASSWORD
    XE="${XE} -s ${SERVER} -u ${USERNAME} -pw ${PASSWORD}"
fi

```

次に、コマンドライン引数をチェックします。複製する仮想マシンの UUID のみが必要です。

```

# Check if there's a VM by the uuid specified
${XE} vm-list params=uuid | grep -q "${vmuuid}"
if [ $? -ne 0 ]; then
    echo "error: no vm uuid \"${vmuuid}\" found"
    exit 2
fi

```

その後、仮想マシンの電源状態を確認し、実行中の場合は完全なシャットダウンを試みます。イベントシステムを使用して、仮想マシンが停止状態 (halted) になるまで待機します。

#### 注：

xe CLI では、コマンドライン引数 *--minimal* がサポートされます。この引数を指定すると、出力内容から余分な空白や書式が削除され、スクリプトからの使用に適した形式が使用されます。複数の値が返される場合は、コンマで区切られます。





```
# Check the power state of the vm
name=$(XEX vm-list uuid=${vmuuid} params=name-label --minimal)
state=$(XEX vm-list uuid=${vmuuid} params=power-state --minimal)
wasrunning=0
```

```
# If the VM state is running, we shutdown the vm first
if [ "${state}" = "running" ]; then
    XEX vm-shutdown uuid=${vmuuid}
    XEX event-wait class=vm power-state=halted uuid=${vmuuid}
    wasrunning=1
fi
```

次に、仮想マシンを複製し、新しい仮想マシンの *name\_label* を *cloned\_vm* に設定します。

```
# Clone the VM
newuuid=$(XEX vm-clone uuid=${vmuuid} new-name-label=cloned_vm)
```

最後に、複製元の仮想マシンが実行中であった場合は、その仮想マシンと新しい仮想マシンの両方を起動します。

```
# If the VM state was running before cloning, we start it again
# along with the new VM.
if [ "$wasrunning" -eq 1 ]; then
    XEX vm-start uuid=${vmuuid}
    XEX vm-start uuid=${newuuid}
fi
```

# 第 5 章 HTTP による XenServer との対話

XenServer では各ホストに HTTP インターフェイスを介してアクセスし、さまざまな操作を実行できます。この章では、利用できるメカニズムについて説明します。

## 5.1. 仮想マシンのインポートとエクスポート

仮想マシンのインポートとエクスポートには時間がかかるので、インポートとエクスポートの操作に対して非同期 HTTP インターフェイスが提供されています。XenServer API を使用してエクスポートするには、次のコードサンプルのように HTTP GET コールを作成して、有効なセッション ID、タスク ID、および仮想マシンの UUID を提供します。

```
task = Task.create()
result = HTTP.get(
    server, 80, "/export?session_id=<session_id>&task_id=<task_id>&ref=<vm_uuid>");
```

インポートするには、次のコードサンプルのように HTTP PUT コールを使用します。

```
task = Task.create()
result = HTTP.put(
    server, 80, "/import?session_id=<session_id>&task_id=<task_id>&ref=<vm_uuid>");
```

## 5.2. XenServer パフォーマンス統計値の取得

XenServer では、XenServer のさまざまな側面のパフォーマンスについて統計値が記録されます。メトリクスは永続的に格納されるため、履歴データに基づく傾向分析や長期間のアクセスが可能です。仮想マシンでストレージを使用できる場合、これらの統計値は仮想マシンのシャットダウン時にディスクに書き込まれます。統計値はラウンドロビンデータベースに格納されます。このデータベースは（コントロールドメインを含む）各仮想マシンおよびサーバーごとに保持されます。ラウンドロビンデータベースは、仮想マシンが実行中の場合はそのホストサーバーに、仮想マシンが実行中でない場合はプールマスタに格納されます。ラウンドロビンデータベースは毎日バックアップされます。

### 警告：

以前のバージョンの XenServer API では、**VM\_metrics**、**VM\_guest\_metrics**、**host\_metrics** の各メソッドおよび関連するメソッドにより瞬間的なパフォーマンスメトリクスを取得できました。これらのメソッドは、この章で説明する HTTP ハンドラを使用するために廃止されました。HTTP ハンドラを使用して、仮想マシンおよびサーバー上のラウンドロビンデータベースから統計値をダウンロードします。デフォルトでは、従来の測定値の戻り値が 0 であることに注意してください。旧バージョンの XenServer の定期的な統計ポーリングの機能を使用するには、`other-config:rrd_update_interval=<interval>` パラメータに次のいずれかの値を設定して、ホストを再起動します。

never これがデフォルト設定で、定期的なポーリングを実行しません。

1 5 秒間隔でポーリングを実行します。

2 1 分間隔でポーリングを実行します。

以前のメトリクス API はデフォルトで値を返さないため、従来の監視プロトコルを使用する監視クライアントを実行するには、このキーを有効にする必要があります。

統計値は最長 1 年間、値が取得された時期に応じて異なる粒度で保持されます。平均および最新の値は次の間隔で保存されます。

- 直近の 10 分間は 5 秒間隔
- 過去 2 時間は 1 分間隔
- 過去 1 週間は 1 時間間隔
- 過去 1 年間は 1 日間隔

ラウンドロビンデータベースは、未圧縮の XML ファイルとしてディスクに保存されます。各ラウンドロビンデータベースのサイズは、200KiB から、1 年分の統計値を格納した時点で約 1.2MiB になります。

### 警告：

ディスク領域の不足などで統計値を書き込めない場合、統計値は失われ、最後に保存されたバージョンのラウンドロビンデータベースが使用されます。

統計値は、HTTP を使用して XML 形式でダウンロードできます (**wget** を使用するなど)。XML 形式については、<http://oss.oetiker.ch/rrdtool/doc/rrddump.en.html> および <http://oss.oetiker.ch/rrdtool/doc/rrdexport.en.html> を参照してください。HTTP 認証はユーザー名とパスワードでも、セッショントークンでも実行できます。パラメータは URL の後に疑問符 (?) を付けて追加し、アンパサンド (&) で区切ります。

ホスト上のすべての仮想マシン統計値の更新を取得するには、次の URL を使用します。

```
http://<username>:<password>@<host>/rrd_updates?start=<secondssinceepoch>
```

この要求により、rrdtool xport コマンドの出力に使用される XML 形式で、クエリ対象のサーバー上の各仮想マシンについてデータが返されます。エクスポートデータのどの列がどの仮想マシンのデータを示すのかを区別できるように、legend フィールドの値の前に仮想マシンの UUID が付加されます。

ホストの更新も取得する場合は、クエリパラメータ `host=true` を使用します。

```
http://<username>:<password>@<host>/rrd_updates?start=<secondssinceepoch>&host=true
```

期間が短くなるにつれて粒度が細かくなります。つまり、要求する統計値の期間が短いほど、詳細な値を取得することになります。

## rrd\_updates の追加パラメータ

`cf=<ave|min|max>` データ統合モードです。

`interval=<interval>` 報告する値の間隔です。

### 注：

デフォルトでは、ave 統計値だけが使用可能です。仮想マシンの min と max の統計値を取得するには、次のコマンドを実行します。

```
xe pool-param-set uuid=<pool_uuid> other-config:create_min_max_in_new_VM_RRDs
```

ホストのすべての統計値を取得するには次のコマンドを実行します。

```
http://<username:password@host>/host_rrd
```

仮想マシンのすべての統計値を取得するには次のコマンドを実行します。

```
http://<username:password@host>/vm_rrd?uuid=<vm_uuid>
```

## 第 6 章 XenServer API 拡張

XenAPI は仮想マシンのライフサイクル管理のための汎用的かつ総合的なインターフェイスで、XenAPI プロバイダで特定の機能（ストレージプロビジョニングやコンソール処理など）を柔軟に実装できます。XenServer に組み込まれている拡張により、XenCenter のインターフェイスで使用する便利な機能が提供されます。これらのメカニズムのしくみについて、この章で説明します。

通常、XenAPI の拡張を使用するには、さまざまなオブジェクトに対して *other-config* マップキーを指定します。このパラメータを使用するということは、XenServer の特定のリリースでその機能がサポートされていることを意味しますが、将来的に長期間サポートされるということではありません。API 化する機能については常時検討していますが、そのためにはインターフェイスの本質が十分に周知されている必要があります。これらの拡張の使用方法について、フィードバックをお寄せください。今後の製品向上のための貴重な資料として、参考にさせていただきます。

### 6.1. 仮想マシンコンソールの転送

ユーザーに対して仮想マシンコンソールを物理マシンのように表示するために、多くの XenAPI のグラフィックインターフェイスで仮想マシンコンソールへのアクセスが必要になります。ゲストの種類、またアクセス先が物理ホストコンソールであるかどうかに応じて、いくつかの種類のコソールを使用できます。

#### コンソールのアクセス

オペレーティングシステム	テキスト	グラフィック	最適化グラフィック
Windows	なし	VNC (API コールを使用)	RDP (ゲストから直接)
Linux	あり (VNC および API コール経由)	なし	VNC (ゲストから直接)
物理ホスト	あり (VNC および API コール経由)	なし	なし

Windows などのハードウェア支援型仮想マシンでは、VNC でグラフィックコンソールを直接提供します。テキストベースのコソールはなく、またグラフィックコンソールを使用するためのゲストネットワーク設定は不要です。ゲストネットワークが確立されている場合は、リモートデスクトップアクセスを設定し、RDP クライアントを使用して直接接続する方が効率的です（これは XenAPI の外部で行う必要があります）。

Linux などの準仮想化仮想マシンでは、ネイティブテキストコンソールを直接提供します。XenServer には **vncterm** と呼ばれるユーティリティが組み込まれており、このテキストベースのコソールをグラフィック VNC に変換できます。このコンソールは、ゲストネットワークがなくても機能します。前述の Windows の場合と同様に、Linux ディストリビューションでよく使用されるのは、ゲストで VNC を設定してゲストのネットワークインターフェイスを介して直接接続する方法です。

物理ホストのコソールは vt100 コソールとしてのみ使用できます。このコンソールには、コントロールドメインで **vncterm** を実行することによって、VNC コソールとして XenAPI を介してアクセスできます。

RFB (Remote Framebuffer) は VNC で使用されるプロトコルで、『[The RFB Protocol](#)』で規定されています。サードパーティの開発者は、自由に使用できる多くの実装を利用して独自の VNC ビューアを開発できます。ビューアがサポートする必要のある最低バージョンは RFB 3.3 です。

### 6.1.1. API による VNC コンソールの取得

VNC コンソールは、ホストエージェントに渡される特殊な URL を介して取得します。API コールは次の順序で呼び出します。

1. クライアントからマスタ/443 : XML-RPC : **Session.login\_with\_password()**。
2. マスタ/443 からクライアント : 後続のコールで使用されるセッションリファレンスを返します。
3. クライアントからマスタ/443 : XML-RPC : **VM.get\_by\_name\_label()**。
4. マスタ/443 からクライアント : 特定の仮想マシンのリファレンスを返します (物理ホストコンソールを取得する場合はコントロールドメインのリファレンス)。
5. クライアントからマスタ/443 : XML-RPC : **VM.get\_consoles()**。
6. マスタ/443 からクライアント : 仮想マシンに関連付けられているコンソールオブジェクトの一覧を返します。
7. クライアントからマスタ/443 : XML-RPC : **VM.get\_location()**。
8. 要求されたコンソールの位置を記述する URI を返します。URI の形式は、<https://192.168.0.1/console?ref=OpaqueRef:c038533a-af99-a0ff-9095-c1159f2dc6a0> のようになります。
9. クライアントから 192.168.0.1 : HTTP CONNECT `"/console?ref=(...)"`

最後の HTTP CONNECT はやや標準から外れています。HTTP/1.1 RFC では、HTTP CONNECT は URL ではなくホストとポートのみを取るべきとされているためです。HTTP 接続が完了したら、その接続を VNC サーバーとして直接使用することができます。追加の HTTP プロトコル操作は不要です。

このスキーマでは、クライアントからコントロールドメインの IP アドレスに直接アクセスする必要があり、NAT (Network Address Translation : ネットワークアドレス変換) デバイスによりアクセスが拒否される場合には正しく機能しません。CLI を使用してクライアントからコンソール URI を取得し、接続状態を調べることができます。

#### CLI でコンソール URI を取得するには

1. 次のコマンドを実行して、仮想マシンの UUID を取得します。

```
xe vm-list params=uuid --minimal name-label=name
```

2. 次のコマンドを実行して、コンソール情報を取得します。

```
xe console-list vm-uuid=uuid
uuid ( RO): 714f388b-31ed-67cb-617b-0276e35155ef
vm-uuid ( RO): 8acb7723-a5f0-5fc5-cd53-9f1e3a7d3069
vm-name-label ( RO): etch
protocol ( RO): RFB
location ( RO): https://192.168.0.1/console?ref=(...)
```

**ping** などのコマンドラインユーティリティを使用して、*location* フィールドの IP アドレスへの接続状態をテストします。

### 6.1.2. Linux 仮想マシンの VNC 転送の無効化

Linux 仮想マシンを作成したり破棄したりするとき、テキストコンソールを VNC に変換する **vncterm** プロセスがホストエージェントによって自動的に管理されます。テキストコンソールに直接アクセスしたい上級ユーザーは、その仮想マシンの VNC 転送を無効にすることができます。その後、テキストコンソールにはコントロールドメインからのみ直接アクセスできるようになり、XenCenter などのグラフィックインターフェイスではその仮想マシンのコンソールを表示できなくなります。

## CLI を使用して Linux VNC コンソールを無効にするには

1. ゲストを起動する前に、仮想マシンレコードに次のパラメータを設定します。

```
xe vm-param-set uuid=uuid other-config:disable_pv_vnc=1
```

2. 仮想マシンを起動します。

3. 次の CLI コマンドを実行して、仮想マシンのドメイン ID を取得します。

```
xe vm-list params=dom-id uuid=<uuid> --minimal
```

4. ホストコンソール上で次のコマンドを実行して、テキストコンソールに直接接続します。

```
/usr/lib/xen/bin/xenconsole <domain_id>
```

この設定は上級者用です。負荷の高い入出力操作にテキストコンソールを使用することは推奨されません。その代わりに、SSH などのネットワークベースの接続方法を使用して、ゲストに接続してください。

## 6.2. 準仮想化 Linux のインストール

Xen 対応のカーネルを起動する必要があるため、準仮想化 Linux ゲストのインストールは複雑で、ハードウェア支援機能を使用するゲストのインストールとは異なります。ただし、エミュレーションのオーバーヘッドがないため、ネイティブに近いインストール速度が得られるという利点があります。XenServer ではさまざまな Linux ディストリビューションのインストールがサポートされるため、ここでは標準的な手順として説明します。

準仮想化 Linux をインストールするために、**eliloader** と呼ばれる特殊なブートローダーがコントロールドメインに組み込まれています。このブートローダーによって仮想マシンレコードのさまざまな *other-config* キーが起動時に読み取られ、ディストリビューション固有のインストール処理が実行されます。

- *install-repository* : 必須キーで、リポジトリのパスです。値は http、https、ftp、または nfs です。ターゲットインストーラと同じ要領で指定しますが、method=などのプレフィックスは含めません。
- *install-vnc* : デフォルトでは false です。インストール中、使用可能であれば VNC を使用します。
- *install-vncpasswd* : デフォルトでは未指定です。使用する VNC パスワードです。ターゲットディストリビューションのコマンドラインで入力可能な場合に指定します。
- *install-round* : デフォルトでは 1 です。現在のブートローダー処理が何回目かを示します。ユーザーはこの値を変更できません。

### 6.2.1. Red Hat Enterprise Linux 4.1/4.4

**eliloader** を使用して起動を 2 回行います。1 回目は、インストーラ `initrd` と `/opt/xensource/packages/files/guest-installer` からのカーネルを返します。2 回目は、仮想マシンから追加のアップデートディスクを削除し、ブートローダーを **pygrub** に切り替えた後、通常の起動を行います。

Red Hat 社ではこれらのディストリビューションに Xen カーネルを提供していないため、この方法で XenServer のカスタムカーネルを代わりに使用する必要があります。

### 6.2.2. Red Hat Enterprise Linux 4.5/5.0

Red Hat Enterprise Linux 4.4 の場合と同様ですが、カーネルと RAM ディスクをユーザー指定のネットワークリポジトリから直接ダウンロードし、ブートローダーを直ちに **pygrub** に切り替える点が異なります。pygrub は直ちに実行されず、次回起動時に解析のみ行われる点に注意してください。

Red Hat ベンダのネットワークリポジトリからアップストリームカーネルを直接インストールすることができます。XenServer に付属の ISO イメージ `xs-tools.iso` では、さまざまな Xen 関連の問題を修正する、アップデートされたカーネルも提供されています。



### 6.2.3. SUSE Linux Enterprise 10 Service Pack 1

このインストールでは2回の起動が必要です。1回目の起動では、ネットワークリポジトリからカーネルとRAMディスクをダウンロードして起動します。2回目の起動では、ディスクを調べてインストールされたカーネルとRAMディスクを見つけ、*PV-bootloader-args* を設定してゲストファイルシステムにこれらのパスを反映させます。このプロセスは、**pygrub** の代わりに SUSE で使用される **domUloader** をエミュレートします。最後に、ブートローダーを **pygrub** に設定し、通常の起動を開始します。

SUSE Linux Enterprise Server 10 のインストール方法では、カーネルとRAMディスクのパスがゲストの *menu.lst* ではなく仮想マシンレコードに保存されます。YaST パッケージマネージャは有効な *menu.lst* を書き込まないため、この方法でインストールする必要があります。

### 6.2.4. CentOS 4.5/5.0

**eliloader** によって認識されるいくつかの MD5 チェックサムが異なることを除き、CentOS のインストール方法は上記の Red Hat の場合と同様です。

## 6.3. Xenstore エントリの仮想マシンへの追加

仮想マシンの種類に応じて特別な操作を行うゲストエージェントを、仮想マシンにインストールしたい場合があります。これを行うには、仮想マシンの作成時に設定される、*vm-data* と呼ばれる特別な Xenstore ネームスペースを使用します。このネームスペースを設定するには、仮想マシンレコードの *xenstore-data* マップを使用します。

### 仮想マシンの Xenstore ノード *foo* を設定するには

1. 次のコマンドを実行して、仮想マシンレコードの *xenstore-data* パラメータを設定します。
 

```
xe vm-param-set uuid=<vm_uuid> xenstore-data:vm-data/foo=bar
```
2. 仮想マシンを起動します。
3. Linux ベースの仮想マシンの場合、XenServer Tools をインストールしてから **xenstore-read** を使用して Xenstore にノードが存在することを確認します。

#### 注：

*vm-data* で始まるプレフィックスのみが許可され、このネームスペースにないものはすべて仮想マシンの起動時に無視されます。

## 6.4. セキュリティの強化

悪意のあるゲストからの攻撃に備えて、XenServer 6.5 以降のコントロールドメインには、さまざまなセキュリティ強化が施されています。この変更により開発済みのアプリケーションの動作に問題が生じることはありませんが、ほかのディストリビューションとは異なる動作として、ここで説明します。

- **libxenstore** を使用してアクセスする、ソケットインターフェイスの **xenstored**。インターフェイスは *xs\_restrict()* によって制限されます。
- **libxenctrl** の *xs\_evtchn\_open()* を呼び出してアクセスする、デバイスの */dev/xen/evtchn*。ハンドルは *xs\_evtchn\_restrict()* を使用して制限できます。
- **libxenctrl** の *xs\_interface\_open()* を呼び出してアクセスする、デバイスの */proc/xen/privcmd*。ハンドルは *xc\_interface\_restrict()* を使用して制限できます。一部の特権コマンド（任意のハイパーコールを呼び出す機能など）はその性質上制限することが難しいため、制限付きハンドルでは使用を禁じられています。
- 制限付きハンドルに後からより高度な権限を付与することはできないため、インターフェイスをいったん終了してから、再開する必要があります。それ以降プロセスでさらにハンドルが開かれる場合は、セキュリティを維持できません。

コントロールドメインの特権ユーザースペースのインターフェイスは、特定のドメインのみを対象に動作するように制限できるようになりました。この変更によって影響を受けるインターフェイスは3つです。

- デバイスエミュレーションの **qemu** プロセスと端末エミュレーションの **vncterm** プロセスは、ルート以外のユーザー ID で実行され、空のディレクトリに制限されます。可能な場合は、上記の制限 API を使用して権限を放棄します。
- 悪意のあるゲストによるコントロールドメインへのサービス拒否攻撃を防ぐため、Xenstore へのアクセスには制限が設定されています。この制限は、トークンバケットという方式で実装されています。ほとんどの操作に 1 トークンが、トランザクションのオープンには 20 トークンが必要です。多数のゲストで同時に負荷の大きな操作を実行するような場合でも上限に達することがないように、十分に高い制限値が設定されています。
- VNC ゲストコンソールは localhost インターフェイスにのみバインドされるので、ユーザーがコントロールドメインのパケットフィルタを無効にしても、外部から VNC ゲストコンソールにアクセスできるようにはなりません。

## 6.5. ネットワークインターフェイスの詳細設定

仮想ネットワークインターフェイスと物理ネットワークインターフェイスにはいくつかの詳細設定があり、*other-config* マップパラメータを使用して設定できます。この詳細設定には、一連のカスタム *ethtool* 設定と、そのほかの設定があります。

### 6.5.1. ethtool 設定

仮想ネットワークインターフェイスと物理ネットワークインターフェイス用にカスタムの *ethtool* 設定が必要な場合があります。この設定は、*other-config* マップパラメーターで *ethtool-<option>* キーを使用して行います。

キー	説明	有効な設定
<i>ethtool-rx</i>	RX チェックサム機能を有効にするかどうかを指定します。	有効にする場合は <i>on</i> または <i>true</i> 、無効にする場合は <i>off</i> または <i>false</i> です。
<i>ethtool-tx</i>	TX チェックサム機能を有効にするかどうかを指定します。	有効にする場合は <i>on</i> または <i>true</i> 、無効にする場合は <i>off</i> または <i>false</i> です。
<i>ethtool-sg</i>	Scatter/Gather を有効にするかどうかを指定します。	有効にする場合は <i>on</i> または <i>true</i> 、無効にする場合は <i>off</i> または <i>false</i> です。
<i>ethtool-tso</i>	TCP セグメンテーションオフロードを有効にするかどうかを指定します。	有効にする場合は <i>on</i> または <i>true</i> 、無効にする場合は <i>off</i> または <i>false</i> です。
<i>ethtool-ufo</i>	UDP フラグメンテーションオフロードを有効にするかどうかを指定します。	有効にする場合は <i>on</i> または <i>true</i> 、無効にする場合は <i>off</i> または <i>false</i> です。
<i>ethtool-gso</i>	汎用セグメンテーションオフロードを有効にするかどうかを指定します。	有効にする場合は <i>on</i> または <i>true</i> 、無効にする場合は <i>off</i> または <i>false</i> です。



キー	説明	有効な設定
ethtool-autoneg	自動ネゴシエーションを有効にするかどうかを指定します。	有効にする場合は on または true、無効にする場合は off または false です。
ethtool-speed	デバイス速度を MB/秒単位で設定します。	10、100、または 1000
ethtool-duplex	全二重モードまたは半二重モードを設定します。	半二重モードにする場合は half、全二重モードにする場合は full です。

たとえば、xe CLI を使用して仮想 NIC の TX チェックサムを有効にするには、次のコマンドを実行します。

```
xe vif-param-set uuid=<VIF UUID> other-config:ethtool-tx="on"
```

または

```
xe vif-param-set uuid=<VIF UUID> other-config:ethtool-tx="true"
```

xe CLI を使用して物理 NIC を半二重モードに設定するには、次のコマンドを実行します。

```
xe vif-param-set uuid=<VIF UUID> other-config:ethtool-duplex="half"
```

### 6.5.2. そのほかの設定

仮想ネットワークインターフェイスまたは物理ネットワークインターフェイスを無作為検出モードに設定することもできます。これを行うには、*promiscuous* キーに on を指定します。たとえば、xe CLI を使用して物理 NIC の無作為検出モードを有効にするには、次のコマンドを実行します。

```
xe pif-param-set uuid=<PIF UUID> other-config:promiscuous="on"
```

または

```
xe pif-param-set uuid=<PIF UUID> other-config:promiscuous="true"
```

VIF オブジェクトと PIF オブジェクトには *MTU* パラメータがあります。このパラメータは読み取り専用で、インターフェイスの最大送信単位の現在の設定を示します。*other-config* マップパラメータで *mtu* キーを使用すると、物理 NIC または仮想 NIC のデフォルトの最大送信単位を上書きすることができます。たとえば、xe CLI を使用してジャンボフレームを使用するように仮想 NIC の MTU をリセットするには、次のコマンドを実行します。

```
xe vif-param-set uuid=<VIF UUID> other-config:mtu=9000
```

使用されるインターフェイスの MTU を変更することは高度で実験的な機能であることと、単一のリソースプール内の複数の NIC 間で MTU が異なると予期しない副作用を起こす可能性があることに注意してください。

## 6.6. ストレージリポジトリ名の国際化対応

インストール時に作成されるストレージリポジトリに、名前を国際化する方法を示す *other\_config* キーが付加されました。

*other\_config["i18n-key"]* は、次のいずれかの値を取ります。

- *local-hotplug-cd*
- *local-hotplug-disk*
- *local-storage*

- *xenserver-tools*

さらに、`other_config["i18n-original-value-<field name>"]`は、ストレージリポジトリの作成時にそのフィールドに設定されていた値を提供します。XenCenterで`SR.name_label`が`other_config["i18n-original-value-name_label"]`と等しいレコードが検出された場合（XenServerのインストール時に作成されてからレコードが変更されていない場合）、国際化が適用されます。つまり、XenCenterでそのフィールドの現在の値が無視され、ユーザー言語に適した値が使用されます。

独自の目的のために`SR.name_label`を変更すると、`other_config["i18n-original-value-name_label"]`と同じではなくなります。したがって、XenCenterで国際化が適用されず、その代わりにユーザーが指定する名前が保持されます。

## 6.7. XenCenterでのオブジェクトの非表示化

ネットワーク、物理ネットワークインターフェイス、および仮想マシンは、`HideFromXenCenter=true`キーをそのオブジェクトの`other_config`パラメータに追加することで、XenCenterで非表示にすることができます。この機能はXenServerに習熟した独立系ソフトウェアベンダのために用意されているもので、日常的なユーザーのためのものではありません。たとえば、環境内の一般ユーザーが直接使用すべきでない、特定の複製仮想マシンを隠す必要がある場合などに使用します。

XenCenterでは、**【表示】**メニューを使用して、隠しオブジェクト（非表示のネットワーク、物理ネットワークインターフェイス、および仮想マシン）を表示することができます。

## 第 7 章 XenCenter API 拡張

この章では、文書化された API とは別に開発された API について、その前提条件と合わせて詳しく説明します。拡張は、VM.other\_config のようなディクショナリに、特定のキー/値ペアとしてコード化されています。

### 7.1. pool

キー	説明
pool.name_label	name_label に値がない場合は、プールをツリー表示から隠す必要があることを示します。
pool.rolling_upgrade_in_progress	プールがローリングアップグレード中である場合に存在します。

### 7.2. Host

キー	説明
host.other_config["iscsi_iqn"]	ホストの iSCSI IQN です。
host.license_params["expiry"]	ホストのライセンスの有効期限です。ISO 8601 の UTC 形式です。
host.license_params["sku_type"]	ホストのライセンスの種類です。Server、Enterprise などです。
host.license_params["restrict_pooling"]	プール化がホストにより制限されている場合は true を返します。
host.license_params["restrict_connection"]	XenCenter から確立できる接続数を制限することを示します。
host.license_params["restrict_qos"]	ホストで QoS (サービス品質) 設定が有効である場合は true を返します。
host.license_params["restrict_vlan"]	ホストで仮想ネットワークの作成が制限されている場合は true を返します。
host.license_params["restrict_pool_attached_storage"]	ホストで共有ストレージの作成が制限されている場合は true を返します。
host.software_version["product_version"]	ホストの製品バージョンを返します。
host.software_version["build_number"]	ホストのビルド番号を返します。
host.software_version["xapi"]	ホストの API リビジョン番号を返します。
host.software_version["package-linux"]	Linux Pack がインストール済みである場合は installed を返します。
host.software_version["oem_build_number"]	ホストが OEM バージョンである場合はそのリビジョン番号を返します。

キー	説明
host.logging["syslog_destination"]	XenServer のシステムログの書き込み先を取得または設定します。ローカルの場合は null です。
host.logging["multipathing"]	ホストでストレージのマルチパスが有効な場合は true です。
host.logging["boot_time"]	浮動小数点 UNIX 時間でのホストの起動時刻です。
host.logging["agent_start_time"]	浮動小数点 UNIX 時間でのコントロールドメイン管理デーモンの開始時刻です。

### 7.3. VM

キー	説明
VM.other_config["default_template"]	テンプレートが Citrix によりインストールされたものであることを示します。この API は、デフォルトテンプレートについてツリー表示から選択的に隠したり、異なるアイコンを使用したり、削除を禁止したりするときに使用します。
VM.other_config["xensource_internal"]	テンプレートが P2V サーバーテンプレートのような特殊なものであることを示します。このようなテンプレートはユーザーインターフェイスに表示されません。
VM.other_config["install_distro"] == "rhlike"	テンプレートが Red Hat Enterprise Linux 4.5、Red Hat Enterprise Linux 5、または CentOS の同等のバージョンのものであることを示します。このテンプレートは、インストール時にインストールリポジトリの入力を要求するため (XenServer 4.1 以降で ISO/CD からのインストールもサポートします)、そしてインストーラに適合するように NFS URL を変更するために使用します。
VM.other_config["install_distro"] in { "rhel41"   "rhel44" }	テンプレートが Red Hat Enterprise Linux 4.1、Red Hat Enterprise Linux 4.4、または CentOS の同等のバージョンのものであることを示します。このテンプレートは、インストール時にインストールリポジトリの入力を要求するため、そしてインストーラに適合するように NFS URL を変更するために使用します。これらのテンプレートでは ISO はサポートされません。

キー	説明
VM.other_config["install_distro"] == "sleslike"	テンプレートが SUSE Linux Enterprise Server 10 および SUSE Linux Enterprise Server 9 のものであることを示します。このテンプレートは、EL5 の場合と同様に、インストール時にインストールリポジトリの入力を要求するために使用します。ただし、NFS URL は変更されません。XenServer 6.5 では、SUSE Linux Enterprise Server 10 のインストールで ISO がサポートされます。このプラットフォームでは、 <i>install-methods</i> を使用して SUSE Linux Enterprise Server 9 と SUSE Linux Enterprise Server 10 を区別します。
VM.other_config["install-repository"] == "cdrom"	URL ではなく、仮想マシンに接続されている CD ドライブ内のリポジトリからのインストールを要求します。
VM.other_config["auto_poweron"]	サーバーの起動時に仮想マシンを起動するかどうかを取得または設定します。true または false です。
VM.other_config["ignore_excessive_vcpus"]	仮想マシンにそのホストが搭載する物理 CPU より多く VCPU が設定されている場合に、XenCenter の警告を無視するかどうかを取得または設定します。無視する場合は true です。
VM.other_config["HideFromXenCenter"]	XenCenter のツリー表示に仮想マシンを表示するかどうかを取得または設定します。隠す場合は true です。
VM.other_config["import_task"]	仮想マシンを作成したインポートタスクを取得します。
VM.HVM_boot_params["order"]	HVM モードの仮想マシンでのみ、仮想マシンの起動順序を取得または設定します。たとえば、DCN は起動順序がディスク、CD ドライブ、ネットワークであることを示します。
VM.VCPU_params["weight"]	仮想マシンの VCPU の <i>ionice</i> 値を取得または設定します。値の範囲は 1 から 65536 で、65536 が上限です。
VM.pool_migrate(..., options['live'])	true はライブマイグレーションを示します。XenCenter では常にこの設定が使用されます。
VM.other_config["install-methods"]	テンプレートで使用できるインストール方法の、コマンド区切りの一覧です。cdrom、nfs、http、または ftp が含まれます。

キー	説明
VM.other_config["last_shutdown_time"]	仮想マシンの最終シャットダウンまたは再起動の日時です。ISO 8601 の UTC 形式です。
VM.other_config["p2v_source_machine"]	仮想マシンが P2V プロセスでインポートされた場合は、インポート元のマシンです。
VM.other_config["p2v_import_date"]	P2V プロセスでインポートされた場合は、仮想マシンのインポート日時です。ISO 8601 の UTC 形式です。

## 7.4. SR

キー	説明
SR.other_config["auto-scan"]	ストレージリポジトリの変更が自動的にスキャンされます。XenCenter で作成されたストレージリポジトリはすべてスキャンされます。
SR.sm_config["type"]	物理 CD ドライブであるストレージリポジトリについて、cd を種類として設定します。

## 7.5. VDI

キー	説明
VDI.type	user の場合は、ディスクが仮想マシンに接続されている場合にグラフィックユーザーインターフェイスで仮想ディスクイメージを削除できます。system の場合は削除できません。後者を使用する目的は、仮想マシンの破損を防ぐことです。仮想ディスクイメージを削除する代わりに、仮想マシンをアンインストールする必要があります。suspend および crashdump では、それぞれ一時停止ダンプとコアダンプが記録されます。ephemeral は現在使用されていません。
VDI.managed	管理されていないすべての仮想ディスクイメージが、ユーザーインターフェイスで非表示になります。これらは、VHD チェーンに分岐点または未使用の LUN-per-VDI ディスクです。
VDI.sm_config["vmhint"]	仮想ディスクイメージでサポートされる仮想マシンの UUID です。特定のストレージバックエンドのパフォーマンスを向上するために、ユーザーインターフェイスを介して仮想ディスクイメージが作成されるときにこの値が設定されます。



## 7.6. VBD

キー	説明
VBD.other_config["is_owner"]	設定されている場合は、仮想マシンのアンインストール時にディスクを削除できます。
VBD.other_config["class"]	<b>ionice</b> の Best Effort 設定に対応する整数を設定します。

## 7.7. network

キー	説明
network.other_config["automatic"]	false 以外の値が設定されている場合、[新規 VM] ウィザードにより、デフォルトでこのネットワークに接続する仮想ネットワークインターフェイスが作成されます。
network.other_config["import_task"]	ネットワークを作成したインポートタスクを取得します。

## 7.8. VM\_guest\_metrics

キー	説明
PV_drivers_version["major"]	仮想マシンの準仮想化ドライバのメジャーバージョンを取得します。
PV_drivers_version["minor"]	仮想マシンの準仮想化ドライバのマイナーバージョンを取得します。
PV_drivers_version["micro"]	仮想マシンの準仮想化ドライバのマイクロバージョン（ビルド番号）を取得します。

## 7.9. Task

キー	説明
task.other_config["object_creation"] "complete"	== 仮想マシンのインポートに関連するタスクについて、すべてのオブジェクト（VM および Network）が作成されるとこのフラグが設定されます。その後、仮想マシンのインポートウィザードですべての必要なネットワークを再マップすると便利です。