Citrix XenServer®7.1 소프트웨어 개발 키트

Publication date March 2017
1.0 엮음

Citrix XenServer®7.1 소프트웨어 개발 키트

버전: 7.1

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

CITRIX

차례

CITRIX®

1장. 소개

XenServer의 개발자 가이드를 시작합니다. 이 문서에는 XenServer에서 제공하는 SDK(소프트웨어 개발 키트)를 이해하고 사용하는 데 필요한 정보가 포함되어 있습니다. 이 정보를 통해 제공된 도구인 API의 기반이 되는 아키텍처에 대한 배경 지식을 얻을 수 있으며 SDK 사용을 빠르게 시작할 수 있는 방법을 확인할 수 있습니다.

CITRIX

2장. 시작

XenServer에는 광범위한 XenServer 관리 기능 및 도구에 대한 프로그래밍 방식 액세스를 제공하는 XML-RPC 기반 API가 포함되어 있습니다. XenServer API는 원격 시스템 및 XenServer 호스트에 대한 로컬에서 호출할 수 있습니다. 원시 XML-RPC 호출을 통해 직접 XenServer Management API를 사용하는 응용 프로그램을 쓸 수는 있지만 개별 API 호출을 대상 언어의 1류 함수로 표시하는 언어 바인딩 사용을 통해 타사 응용 프로그램을 매우 간단하게 개발할 수 있습니다. XenServer SDK에서는 C, C#, Java, Python 및 PowerShell 프로그래밍 언어에 대한 언어 바인딩과 예제 코드를 제공합니다.

2.1. 시스템 요구 사항 및 준비

SDK를 사용하려면 먼저 XenServer를 설치해야 합니다. 무료 버전인 Citrix XenServer는 http://www.citrix.com/downloads/xenserver/에서 다운로드할 수 있습니다. 개발 호스트를 설치하는 방법에 대한 자세한 내용은 XenServer 설치 가이드를 참조하십시오. 설치가 완료되면 호스트 IP 주소와 호스트 암호를 기록해 두십시오.

2.2. 다운로드

ZIP 파일로 패키지된 SDK는 http://www.citrix.com/downloads/xenserver/에서 무료로 다운로드할 수 있습니다.

2.3. SDK 언어

SDK ZIP 파일의 추출된 내용은 XenServer-SDK 디렉터리에 있습니다. 이러한 구조에 대한 개요는 다음과 같습니다. 필요한 경우 하위 디렉터리에 자체 개별 추가 정보 파일이 포함되어 있습니다. 제공된 예제는 모든 언어 바인딩에서 동일하지 않으므로 한 가지 바인딩을 사용하려는 경우 다른 바인딩에서 사용할 수 있는 샘플 코드도 찾아보는 것이 좋습니다.

XenServer-SDK 디렉터리의 최상위 수준에는 API 의미 체계를 더 자세히 설명하고 네트워크상의 XML/RPC 메시지 형식에 대해 설명하는 XenServer API 참조 문서가 포함되어 있습니다.

2.3.1. C

XenServer-SDK 디렉터리에 C 프로그래머와 관련된 다음 폴더가 포함됩니다.

- libxenserver

  C용 XenServer SDK

  - libxenserver/bin

    libxenserver 컴파일된 바이너리.

  - libxenserver/src

    libxenserver 소스 코드 및 예제와 예제 빌드에 필요한 메이크파일(Makefile) 모든 API 개체는 해당 개체의 모든 API 함수에 대한 선언이 포함된 헤더 파일에 연결되어 있습니다. 예를 들어 VM 작업을 호출하는 데 필요한 형식 정의 및 함수는 모두 xen_vm.h에 포함되어 있습니다.

C 바인딩 종속성

지원되는 플랫폼:	Linux 및 Windows(cygwin에서)

라이브러리:	언어 바인딩은 C 프로그램에 의해 연결되는 libxenserver.so로 생성됩니다.
종속성:	• XML 라이브러리(GNU Linux의 libxml2.so) • Curl 라이브러리(libcurl3.so)

다음의 간단한 예제가 C 바인딩과 함께 포함되어 있습니다.

- test_vm_async_migrate: 비동기 API 호출을 사용하여 실행 중인 VM을 슬레이브 호스트에서 풀 마스터로 마이그레이션하는 방법을 보여 줍니다.
- test_vm_ops: 호스트의 기능을 쿼리하고, VM을 만들고, 비어 있는 새 디스크 이미지를 VM에 연결한 다음 다양한 전원 주기 작업을 수행하는 방법을 보여 줍니다.
- test_failures: 오류 문자열을 enum_xen_api_failure로 변환하고 반대로 변환하는 방법을 보여 줍니다.
- test_event_handling: 연결 시 이벤트를 수신하는 방법을 보여 줍니다.
- test_enumerate: 다양한 API 개체를 에뮬레이션하는 방법을 보여 줍니다.
- test_get_records: 호스트, VM 및 스토리지 저장소와 같은 API 개체에 대한 정보를 가져오는 방법을 보여 줍니다.

2.3.2. C#

XenServer-SDK 디렉터리에 C# 프로그래머와 관련된 다음 폴더가 포함됩니다.

- XenServer.NET

  C#.NET용 XenServer SDK

  - XenServer.NET/bin

    XenServer.NET에서 사용할 수 있는 컴파일된 바이너리.

  - XenServer.NET/samples

    Microsoft Visual Studio 솔루션으로 제공되는 XenServer.NET 예제.

  - XenServer.NET/src

    Microsoft Visual Studio 솔루션으로 제공되는 XenServer.NET 소스 코드. 모든 API 개체는 하나의 C# 파일과 연결되어 있습니다. 예를 들어 VM 작업을 구현하는 함수는 VM.cs 파일에 포함되어 있습니다.

C# 바인딩 종속성

지원되는 플랫폼:	.NET 버전 4.5이 설치된 Windows
라이브러리:	언어 바인딩은 C# 프로그램에 의해 참조될 수 있는 동적 연결 라이브러리인 XenServer.dll로 생성됩니다.
종속성:	CookComputing.XMLRpcV2.dll은 XenServer.dll이 xml-rpc 서버와 통신하는 데 필요합니다. 패치가 적용된 2.5 버전이 출시되었으며 이 버전을 사용하는 것이 좋습니다. 하지만 다른 버전도 작동할 수 있습니다.

XenServer-SDK/XneServer.NET/samples 디렉터리에는 XenSdkSample.sln 솔루션의 개별 프로젝트로 다음 세 개의 예제가 C# 바인딩과 함께 포함되어 있습니다.

- GetVariousRecords: XenServer 호스트에 로그인하고 호스트, 스토리지 및 가상 컴퓨터에 대한 정보를 표시합니다.
- GetVmRecords: XenServer 호스트에 로그인하고 모든 VM 레코드를 나열합니다.
- VmPowerStates: XenServer 호스트에 로그인하고 VM을 찾고 다양한 전원 상태를 수행합니다. 종료할 VM이 이미 설치되어 있어야 합니다.

2.3.3. Java

XenServer-SDK 디렉터리에 C# 프로그래머와 관련된 다음 폴더가 포함됩니다.

- XenServerJava

  Java용 XenServer SDK

  - XenServerJava/bin

    Java 컴파일된 바이너리.

  - XenServerJava/javadoc

    Java 설명서

  - XenServerJava/samples

    Java 예제

  - XenServerJava/src

    Java 소스 코드와 코드 및 예제 빌드에 필요한 메이크파일(Makefile) 모든 API 개체는 하나의 Java 파일과 연결되어 있습니다. 예를 들어 VM 작업을 구현하는 함수는 VM.java 파일에 포함되어 있습니다.

Java 바인딩 종속성

지원되는 플랫폼:	Linux 및 Windows
라이브러리:	언어 바인딩은 Java 프로그램에 의해 연결되는 Java 보관 파일 xenserver-7.1.jar로 생성됩니다.
종속성:	• xmlrpc-client-3.1.jar 및 xmlrpc-common-3.1.jar은 xenserver.jar이 xml-rpc 서버와 통신하는 데 필요합니다. • ws-commons-util-1.0.2.jar은 예제를 실행하는 데 필요합니다.

기본 파일 XenServer-SDK/XenServerJava/samples/RunTests.java를 실행하면 동일한 디렉터리에 포함된 일련의 예제가 실행됩니다.

- AddNetwork: NIC에 연결되지 않은 새 내부 네트워크를 추가합니다.
- SessionReuse: 여러 연결에서 세션 개체를 공유하는 방법을 보여 줍니다.
- AsyncVMCreate: 기본 제공 템플릿에서 새 VM을 비동기적으로 만들고 VM을 시작 및 중지합니다.
- VdiAndSrOps: 더미 SR 만들기를 포함하여 다양한 SR 및 VDI 테스트를 수행합니다.
- CreateVM: 네트워크 및 DVD 드라이브가 있는 기본 SR에서 VM을 만듭니다.
- DeprecatedMethod: 사용되지 않는 API 방법이 호출될 경우 경고가 표시되는지 테스트합니다.
- GetAllRecordsOfAllTypes: 모든 레코드에서 모든 개체 유형을 검색합니다.

- SharedStorage: 공유 NFS SR을 만듭니다.
- StartAllVMs: 호스트에 연결하고 호스트의 각 VM을 시작하려고 시도합니다.

2.3.4. PowerShell

XenServer-SDK 디렉터리에 PowerShell 사용자와 관련된 다음 폴더가 포함됩니다.

- XenServerPowerShell

  PowerShell용 XenServer SDK입니다.

  - XenServerPowerShell/XenServerPSModule

    XenServer PowerShell 모듈입니다.

  - XenServerPowerShell/samples

    PowerShell 예제 스크립트입니다.

  - XenServerPowerShell/src

    XenServer PowerShell cmdlet에 대한 C# 소스 코드.

자세한 설치 지침은 모듈과 함께 제공되는 README 파일 내에 제공됩니다. 모듈을 설치한 후 다음을 입력하여 cmdlet의 개요를 가져올 수 있습니다.

```
PS> Get-Help about_XenServer
```

PowerShell 바인딩 종속성

지원되는 플랫폼:	.NET Framework 4.5 및 PowerlShell v4.0이 설치된 Windows
라이브러리:	XenServerPSModule
종속성:	CookComputing.XMLRpcV2.dll은 xml-rpc 서버와 통신하는 데 필요합니다. 패치가 적용된 2.5 버전이 출시되었으며 이 버전을 사용하는 것이 좋습니다. 하지만 다른 버전도 작동할 수 있습니다.

다음 예제 스크립트는 XenServer-SDK/XenServerPowerShell/samples 디렉터리에 PowerShell 바인딩과 함께 포함되어 있습니다.

- AutomatedTestCore.ps1: XenServer 호스트에 로그인하고, 스토리지 저장소 및 VM을 만든 다음 다양한 전원 주기 작업을 수행하는 방법을 보여 줍니다.
- HttpTest.ps1: XenServer 호스트에 로그인하고 VM을 만든 다음 VM 가져오기와 내보내기, 패치 업로드, 성능 통계 검색 등과 같은 작업을 수행하는 방법을 보여 줍니다.

2.3.5. Python

XenServer-SDK 디렉터리에 Python 개발자와 관련된 다음 폴더가 포함됩니다.

- XenServerPython

  이 디렉터리에는 XenServer Python 모듈(XenAPI.py)이 포함되어 있습니다. 예제에는 provision.py 라이브러리가 사용됩니다.

  - XenServerPython/samples

Python을 사용하는 XenServer API 예제

Python 바인딩 종속성

지원되는 플랫폼:	Linux
라이브러리:	XenAPI.py
종속성:	없음

SDK에는 7가지 python 예제가 포함되어 있습니다.

- fixpbds.py - 공유 스토리지에 액세스하는 데 사용되는 설정을 다시 구성합니다.
- install.py - Debian VM을 설치하고, 네트워크에 연결하고, 시작한 다음 해당 IP 주소를 보고할 때까지 대기합니다.
- license.py - 새 라이센스를 XenServer 호스트에 업로드합니다.
- permute.py - VM 집합을 선택하고 XenMotion을 사용하여 호스트 사이에서 VM 집합을 동시에 이동합니다.
- powercycle.py - VM 집합을 선택하고 전원 주기 작업을 실행합니다.
- shell.py - 테스트 용도로 사용되는 간단한 대화형 셸(shell)입니다.
- vm_start_async.py - 작업을 비동기로 호출하는 방법을 보여 줍니다.
- watch-all-events.py - 모든 이벤트에 등록하고 이벤트 발생 시 세부 정보를 출력합니다.

2.4. CLI(명령줄 인터페이스)

원시 XML-RPC 또는 제공된 언어 바인딩 중 하나를 사용하는 대신 타사 소프트웨어 개발자는 XE CLI(명령줄 인터페이스) xe를 사용하여 XenServer 호스트와 통합할 수 있습니다. xe CLI는 기본적으로 XenServer 호스트에 설치되며 Linux에서는 독립 실행형 원격 CLI도 사용할 수 있습니다. Windows에서는 xe.exe CLI 실행 파일이 XenCenter와 함께 설치됩니다.

CLI 종속성

지원되는 플랫폼:	Linux 및 Windows
라이브러리:	없음
바이너리:	xe(Windows의 경우 xe.exe)
종속성:	없음

CLI에서는 필요한 세션 관리가 자동으로 수행되므로 거의 모든 API 호출을 스크립트 또는 기타 프로그램에서 직접 호출할 수 있습니다. XE CLI 구문 및 기능은 XenServer 관리자 가이드에 자세하게 설명되어 있습니다. 추가 리소스 및 예제를 보려면 Citrix Knowledge Center를 방문하십시오.

참고

XenServer 호스트 콘솔에서 CLI를 실행하는 경우 명령 이름과 인수 모두에 대해 Tab 완성 기능을 사용할 수 있습니다.

CITRIX

3장. XenServer API 개요

이 장에서는 XenServer API(이하 "API") 및 관련 개체 모델을 소개합니다. API에는 다음과 같은 주요 기능이 있습니다.

- XenServer 호스트의 모든 영역 관리. API를 사용하면 VM, 스토리지, 네트워킹, 호스트 구성 및 풀을 관리할 수 있습니다. 성능 및 상태 메트릭도 API를 통해 쿼리할 수 있습니다.

- 영구 개체 모델. 파생 작업을 생성하는 모든 작업(예: 개체 만들기, 삭제 및 매개 변수 수정)의 결과는 XenServer 설치에서 관리하는 서버 측 데이터베이스에 유지됩니다.

- 이벤트 메커니즘. API를 통해 클라이언트는 영구(서버 측) 개체 수정 시 알림을 받도록 등록할 수 있습니다. 이렇게 하면 응용 프로그램이 동시에 실행되는 클라이언트에서 수행한 데이터 모델 수정 내용을 추적할 수 있습니다.

- 동기 및 비동기 호출. 모든 API 호출은 동기적으로 호출될 수 있습니다. 즉, 호출이 완료될 때까지 다른 호출은 차단됩니다. 오래 실행될 수 있는 API 호출은 비동기적으로 호출될 수도 있습니다. 비동기 호출은 작업 개체에 대한 참조와 함께 즉시 반환됩니다. API를 통해 이 작업 개체를 쿼리하여 진행률과 상태 정보를 얻을 수 있습니다. 비동기적으로 호출된 작업이 완료되면 작업 개체를 통해 결과(또는 오류 코드)를 확인할 수 있습니다.

- 원격으로 사용 가능 및 크로스 플랫폼. API 호출을 실행하는 클라이언트는 관리되는 호스트에 있지 않아도 되며 API를 실행하기 위해 ssh를 통해 호스트에 연결되지 않아도 됩니다. API 호출은 XML-RPC 프로토콜을 사용하여 네트워크를 통해 요청 및 응답을 전송합니다.

- 안전하고 인증된 액세스. 호스트에서 실행되는 XML-RPC API 서버는 보안 소켓 연결을 허용하므로 클라이언트는 https 프로토콜을 통해 API를 실행할 수 있습니다. 또한 모든 API 호출은 서버의 사용자 이름 및 암호 확인을 통해 생성된 로그인 세션 컨텍스트에서 실행됩니다. 따라서 XenServer 설치에 대해 안전하고 인증된 액세스가 제공됩니다.

3.1. API 시작

XenServer 설치에서 새 VM을 만들고 이에 대해 시작/일시 중단/다시 시작/중지 주기를 수행하는 호출을 설명함으로써 API에 대한 설명을 시작합니다. 여기서는 특정 언어로 작성된 코드를 참조하지 않고 "설치 및 시작" 작업을 수행하는 RPC 호출 시퀀스에 대해 자유롭게 설명합니다.

참고

향후 API 버전에서 제거되거나 변경될 수 있는 VM.create 호출을 사용하지 않는 것이 좋습니다. 새 VM을 만드는 다른 방법에 대해 알아보려면 계속 읽어보시기 바랍니다.

3.1.1. 인증: 세션 참조 가져오기

첫 번째 단계는 Session.login_with_password(<username>, <password>, <client_API_version>, <originator>)를 호출하는 것입니다. API는 세션을 기반으로 하므로 다른 호출을 수행하기 전에 서버에 대한 인증을 수행해야 합니다. 사용자 이름 및 암호가 올바르게 인증되었다고 가정하는 경우 이 호출을 수행하면 세션 참조를 가져올 수 있습니다. 이후 API 호출은 이 세션 참조를 매개 변수로 사용합니다. 이 방법으로 적절한 권한을 부여받은 API 사용자만 XenServer 설치에서 작업을 수행할 수 있도록 할 수 있습니다. 모든 API

호출에 같은 세션을 계속 사용할 수 있습니다. Citrix에서는 세션이 끝나면 Session.logout(session)을 호출하여 정리할 것을 권장합니다. 자세한 내용은 뒷부분을 참조하십시오.

3.1.2. 새 VM 설치의 기반이 되는 템플릿 목록 가져오기

다음 단계는 호스트에서 "템플릿" 목록을 쿼리하는 것입니다. 템플릿은 특별하게 표시된 VM 개체로, 지원되는 다양한 게스트 유형에 적합한 기본 매개 변수를 지정합니다. xe template-list CLI 명령을 실행하면 XenServer 설치의 템플릿 목록을 직접 간단하게 확인할 수 있습니다. API에서 템플릿 목록을 가져오려면 서버에서 is_a_template 필드가 true로 설정된 VM 개체를 찾아야 합니다. 이를 수행하는 한 가지 방법은 앞의 호출에서 가져온 참조를 session 매개 변수로 사용하여 를 호출하는 것입니다. 이 호출은 서버를 쿼리하여 모든 VM 개체 참조와 해당 필드 값을 포함하는 호출 당시 생성된 스냅샷을 반환합니다.

이 단계에서는 반환된 개체 참조 및 필드 값을 특정 클라이언트 언어로 조작할 수 있는 메커니즘에 대해서는 다루지 않습니다. 세부적인 내용은 언어별 API 바인딩에서 다루고 다음 장에서 구체적으로 설명합니다. 지금은 API 호출에서 반환된 개체 및 필드 값을 읽고 조작하는 추상적인 메커니즘이 있다고 가정하는 것으로 충분합니다.

이제 클라이언트 응용 프로그램의 메모리에 모든 VM 개체 필드 값에 대한 스냅샷이 들어 있으므로 작업을 반복하여 "is_a_template"이 true로 설정된 개체를 찾기만 하면 됩니다. 이 단계에서는 예제 응용 프로그램이 템플릿 개체에 대한 작업을 반복할 뿐만 아니라 해당 "name_label"이 "Debian Etch 4.0"(XenServer와 함께 제공되는 기본 Linux 템플릿 중 하나)으로 설정된 개체에 해당하는 참조를 기억한다고 가정합니다.

3.1.3. 템플릿을 기반으로 VM 설치

제공된 예제를 계속 사용하여 이제 선택한 템플릿을 기반으로 새 VM을 설치해야 합니다. 설치 프로세스에는 두 번의 API 호출이 필요합니다.

- 먼저 API 호출 VM.clone(session, t_ref, "my first VM")을 수행해야 합니다. 이 호출에서는 서버가 새 VM 개체를 만들기 위해 t_ref가 참조하는 VM 개체를 복제하도록 지시합니다. 이 호출의 반환 값은 새로 생성된 VM에 해당하는 VM 참조입니다. 이 new_vm_ref를 호출해 봅니다.
- 이 단계에서 t_ref가 참조하는 복제 원본인 VM 개체와 마찬가지로 new_vm_ref가 참조하는 개체도 역시 템플릿입니다. 를 VM 개체로 만들려면 을 호출해야 합니다. 이 호출이 반환할 때 new_vm_ref 개체의 is_a_template 필드는 false로 설정되어 있는데 이는 new_vm_ref가 시작할 준비가 된 일반 VM을 참조하고 있음을 나타냅니다.

참고

이 호출을 수행하는 동안 템플릿의 디스크 이미지가 만들어지므로 프로비전 작업은 몇 분 정도 걸릴 수 있습니다. Debian 템플릿의 경우 새로 생성된 디스크도 이 단계에서 Debian 루트 파일 시스템으로 채워집니다.

3.1.4. VM에 대해 시작/일시 중단/다시 시작/중지 주기 수행

이제 새로 설치된 VM을 나타내는 개체 참조를 얻었으므로 몇 가지 수명주기 작업을 쉽게 수행할 수 있습니다.

- VM을 시작하려면 VM.start(session, new_vm_ref)를 호출합니다.
- VM이 실행되면 VM.suspend(session, new_vm_ref)를 호출하여 VM을 일시 중단할 수 있습니다.
- 그런 다음 VM.resume(session, new_vm_ref)을 호출하여 다시 시작합니다.
- VM.shutdown(session, new_vm_ref)을 호출하여 VM을 완전히 종료할 수 있습니다.

3.1.5. 로그아웃

응용 프로그램이 XenServer 호스트와 상호 작용을 마치면 Session.logout(session)을 호출하는 것이 좋습니다. 이렇게 하면 세션 참조가 무효화되어 이후 API 호출에 사용할 수 없게 되며 동시에 세션 개체를 저장하는 데 사용된 서버 측 메모리의 할당이 해제됩니다.

비활성 세션은 결국 시간 초과로 종료되지만 서버에는 username 또는 originator 각각에 동시 세션 수 제한이 500으로 하드코딩되어 있습니다. 이 제한에 도달하면 새 로그인을 수행할 때 가장 오래 전에 사용된 세션 개체가 제거되기 때문에 관련 세션 참조도 무효화됩니다. 서버에 동시에 액세스하는 다른 응용 프로그램과 성공적으로 상호 운영되게 하려면 다음 정책을 수행하는 것이 좋습니다.

- 사용할 응용 프로그램과 버전을 식별하는 문자열을 선택합니다.
- 하루를 시작할 때 originator 매개 변수의 확인 문자열로 Session.login_with_password를 사용하여 단일 세션을 만듭니다.
- 이 세션을 응용 프로그램 전체에 사용한 후 가능할 때 명시적으로 로그아웃합니다. 참고로 세션은 여러 개의 개별 클라이언트-서버 네트워크 연결에서 사용할 수 있습니다.

잘못 작성된 클라이언트로 인해 세션 누수가 발생하거나 제한을 초과하는 경우에도 클라이언트가 적절한 originator 인수를 사용하면 XenServer 로그에서 쉽게 식별될 수 있으므로 XenServer가 악성 클라이언트의 가장 긴 유휴 세션만 삭제합니다. 이 경우 해당 클라이언트에게 문제가 발생할 수 있지만 다른 클라이언트에게는 아무 문제도 발생하지 않습니다. 오작동하는 클라이언트가 originator를 지정하지 않은 경우에는 식별이 쉽지 않아 originator를 지정하지 않은 다른 클라이언트의 세션도 중간에 삭제됩니다.

3.1.6. 설치 및 시작 예제: 요약

API를 사용하여 XenServer 템플릿에서 VM을 설치하고 VM에 대한 다양한 수명주기 작업을 수행하는 방법을 살펴보았습니다. 이러한 작업에 영향을 주기 위해 실행해야 한 호출 수는 그리 많지 않았습니다.

- 세션을 얻기 위한 한 번의 호출: Session.login_with_password()
- XenServer 설치에 있는 VM(및 템플릿) 개체를 쿼리하는 한 번의 호출: . 이 호출에서 반환된 정보를 사용하여 설치 원본으로 사용할 적합한 템플릿을 선택했습니다.
- 선택한 템플릿에서 VM을 설치하기 위한 두 번의 호출: VM.clone() 이후 VM.provision() 수행.
- 결과 VM을 시작하기 위한 한 번의 호출: VM.start()(일시 중단, 다시 시작 및 종료하기 위한 유사한 다른 한 번의 호출)
- 로그아웃하기 위한 호출 하나: Session.logout()

API가 전체적으로는 복잡하고 모든 기능을 포함하고 있을지라도 VM을 만들고 VM에 대해 수명주기 작업을 수행하는 등의 일반적인 작업은 간단한 API 호출 몇 개만으로 수행할 수 있는 매우 간단한 작업이라는 것을 알 수 있습니다. 처음 볼 때는 조금 어려워 보일 수도 있는 다음 섹션을 학습하는 동안 이 점을 기억하시기 바랍니다.

3.2. 개체 모델 개요

이 섹션에서는 API의 개체 모델에 대한 높은 수준의 개요를 제공합니다. 여기서 개략적으로 설명한 각 클래스의 매개 변수 및 메서드에 대한 자세한 설명은 XenServer API 참조 문서에서 확인할 수 있습니다.

먼저 API를 구성하는 일부 핵심 클래스에 대해 간략하게 살펴봅니다. 이에 대한 정의가 다소 추상적으로 보이더라도 걱정하지 마십시오. 다음 섹션의 텍스트 설명 및 다음 장의 코드 샘플을 살펴봄으로써 이러한 개념을 더 구체적으로 파악할 수 있습니다.

VM	VM 개체는 XenServer 호스트 또는 리소스 풀의 특정 가상 컴퓨터 인스턴스를 나타냅니다. 이러한 메서드의 예로는 start, suspend, pool_migrate 등이 있으며 이러한 매개 변수의 예로는 power_state, memory_static_max 및 name_label이 있습니다. 이전 섹션에서 VM 클래스를 사용하여 템플릿과 일반 VM을 모두 나타내는 방법에 대해 살펴보았습니다.
호스트	호스트 개체는 XenServer 풀의 물리적 호스트를 나타냅니다. 메서드 예로는 및 이 있으며 매개 변수의 예로는 , 및 [IP] 가 있습니다.
VDI	VDI 개체는 가상 디스크 이미지를 나타냅니다. 가상 디스크 이미지를 VM에 연결할 수 있는데 이렇게 하면 VM 내부에 블록 장치가 표시되고 이를 통해 가상 디스크 이미지로 캡슐화된 비트를 읽고 쓸 수 있습니다. VDI 클래스의 메서드 예로는 "resize" 및 "clone"이 있으며 필드의 예로는 "virtual_size" 및 "sharable"이 있습니다. 앞의 예에서 VM 템플릿에 대해 VM.provision을 호출했을 때 새로 생성된 디스크를 나타내기 위해 일부 VDI 개체가 자동으로 생성되어 VM 개체에 연결되었습니다.
SR	SR(스토리지 저장소)은 VDI 컬렉션을 모으고 VDI의 비트가 저장되는 물리적 스토리지의 속성을 캡슐화합니다. 매개 변수의 예로는 (이에 따라 XenServer 설치에서 SR의 VDI를 읽고 쓰는 데 사용하는 스토리지별 드라이버가 결정됨) 및 이 있으며 메서드의 예로는 (스토리지별 드라이버를 호출하여 SR에 포함된 VDI 및 이 VDI의 속성 목록을 가져옴) 및 (VDI를 저장할 수 있도록 물리적 스토리지 블록을 초기화함)가 있습니다.
네트워킹	네트워크 개체는 XenServer 호스트 인스턴스가 활성화된 환경에 존재하는 계층 2 네트워크를 나타냅니다. XenServer에서는 네트워크를 직접 관리하지 않으므로 이 클래스는 물리적 및 가상 네트워크 토폴로지를 모델링하는 역할만 수행하는 가벼운 클래스입니다. VIF 및 PIF 인스턴스를 통해(아래 참조) 특정 네트워크 개체에 연결된 VM 및 호스트 개체는 서로에게 네트워크 패킷을 보낼 수 있습니다.

이 시점에서 이러한 클래스의 설명이 지루하다고 느끼는 독자는 다음 장의 코드 설명으로 건너뛸 수 있습니다. 이미 설명한 일부 클래스만 사용해도 많은 유용한 응용 프로그램을 작성할 수 있습니다. 클래스에 대한 이론적인 설명을 더 보려는 독자는 계속 읽으십시오.

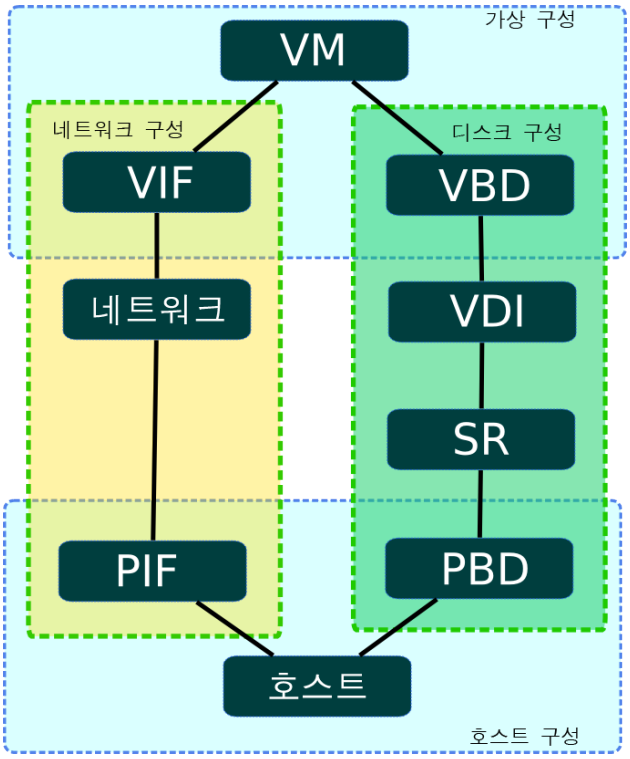
위에 나열된 클래스 외에 VM과 호스트, 스토리지와 네트워크 간 관계를 지정하는 커넥터 역할을 수행하는 클래스가 4개 더 있습니다. 이 클래스 중 처음 두 클래스 VBD 및 VIF는 각각 VM을 가상 디스크와 네트워크 개체에 연결하는 방법을 결정합니다.

VBD	VBD(가상 블록 장치) 개체는 VM과 VDI 간 연결을 나타냅니다. VM이 부팅되면 연결되어야 하는 디스크 이미지(VDI)를 결정하기 위해 해당 VBD 개체가 쿼리됩니다. VBD 클래스 메서드의 예로는 "plug"(디스크 장치를 실행 중인 VM에 핫 플러그하여 지정된 VDI를 액세스 가능하게 함) 및 "unplug"(실행 중인 게스트에서 디스크 장치를 핫 언플러그함)가 있으며 필드의 예로는 "device"(지정된 VDI를 액세스 가능하도록 만드는 게스트 내부의 장치 이름을 결정함)가 있습니다.
VIF	VIF(가상 네트워크 인터페이스) 개체는 VM과 네트워크 개체 간 연결을 나타냅니다. VM이 부팅되면 생성되어야 할 네트워크 장치를 결정하기 위해 VIF 개체가 쿼리됩니다. VIF 클래스 메서드의 예로는 "plug"(실행 중인 VM으로 네트워크 장치를 핫 플러그함) 및 "unplug"(실행 중인 게스트에서 네트워크 장치를 핫 언플러그함)가 있습니다.

두 번째 "커넥터 클래스" 집합은 호스트를 네트워크 및 스토리지에 연결하는 방법을 결정합니다.

PIF	PIF(물리적 인터페이스) 개체는 호스트와 네트워크 개체 간 연결을 나타냅니다. 호스트가 PIF를 통해 네트워크에 연결되어 있는 경우 지정된 호스트의 패킷은 해당 호스트에 의해 전송/수신될 수 있습니다. PIF 클래스 필드의 예로는 "device"(PIF에 해당하는 장치 이름을 지정함 - 예: eth0) 및 "MAC"(PIF가 나타내는 기본 NIC의 MAC 주소를 지정함)이 있습니다. PIF는 물리적 인터페이스와 VLAN을 모두 추상화합니다. 여기서 VLAN은 "VLAN" 필드에서 양의 정수로 구분됩니다.
PBD	PBD(물리적 블록 장치) 개체는 호스트와 SR(스토리지 저장소) 개체 간 연결을 나타냅니다. 필드에는 "currently-attached"(지정된 SR 개체가 나타내는 스토리지 청크를 현재 호스트에서 사용할 수 있는지 여부를 지정함) 및 "device_config"(지정된 호스트에서 하위 수준 스토리지 장치를 구성하는 방법을 결정하는 스토리지 드라이버별 매개 변수를 지정함. 예를 들어 NFS 파일러에 렌더링된 SR의 경우 device_config가 파일러의 호스트 이름 및 SR 파일이 있는 파일러의 경로를 지정할 수 있음)가 있습니다.

그림 3.1. 공통 API 클래스:



VM, 호스트, 스토리지 및 네트워킹 관리용 API 클래스의 그래픽 개요

그림 3.1. “공통 API 클래스”에서는 VM, 호스트, 스토리지 및 네트워킹 관리와 관련된 API 클래스의 그래픽 개요를 보여 줍니다. 이 다이어그램에서 스토리지와 네트워크 구성, 또한 가상 컴퓨터와 호스트 구성이 대칭을 이루는 것을 쉽게 확인할 수 있습니다.

3.3. VIF 및 VBD 사용

이 섹션에서는 좀 더 복잡한 시나리오를 살펴보면서 API를 사용하여 가상 스토리지 및 네트워크 장치와 관련된 다양한 작업을 수행할 수 있는 방법을 자유롭게 설명합니다.

3.3.1. 디스크를 만들어 VM에 연결

먼저 비어 있는 새 디스크 이미지를 만들어 실행 중인 VM에 연결하는 방법을 알아보겠습니다. 이미 실행 중인 VM이 있으며 해당 API 개체 참조를 알고 있다고 가정합니다. 예를 들어 이전 섹션에서 설명한 절차를 사용하여 VM을 만들었으며 서버가 해당 참조를 반환했을 수 있습니다. 또한 XenServer 설치에서 인증되었으며 해당 session reference(세션 참조)를 가지고 있다고 가정합니다. 내용의 간결성을 위해 이 장의 나머지에서는 더 이상 세션에 대해 언급하지 않겠습니다.

3.3.1.1. 비어 있는 새 디스크 이미지 만들기

먼저 물리적 스토리지에서 디스크 이미지를 인스턴스화합니다. 이 작업은 를 호출하여 수행합니다. VDI.create 호출에서는 다음과 같은 여러 매개 변수를 사용합니다.

- name_label 및 name_description: 사람이 쉽게 읽을 수 있는 디스크 이름/설명으로 UI에 편리하게 표시하기 위한 용도 등으로 사용됩니다. 원하는 경우 이러한 필드를 비워 둘 수 있습니다.
- SR: VDI의 비트가 있는 물리적 스토리지를 나타내는 스토리지 저장소에 대한 개체 참조입니다.
- read_only: 이 필드를 true로 설정하면 VDI를 읽기 전용으로만 VM에 연결할 수 있습니다. 필드가 true로 설정되어 있는 VDI를 읽기/쓰기용으로 연결하려고 하면 오류가 발생합니다.

VDI.create를 호출하면 XenServer 설치에서 물리적 스토리지에 빈 디스크 이미지를 만들고, 연결된 VDI 개체(물리적 스토리지의 디스크 이미지를 참조하는 데이터 모델 인스턴스)를 만든 다음 새로 만든 VDI 개체에 대한 참조를 반환합니다.

물리적 스토리지에 디스크 이미지가 표시되는 방식은 만들어진 VDI가 저장된 SR의 유형에 따라 다릅니다. 예를 들어 SR의 유형이 "lvm"이면 새 디스크 이미지는 LVM 볼륨으로 렌더링되고, SR 유형이 "nfs"면 새 디스크 이미지가 NFS 파일러에 만들어진 스파스 VHD 파일로 표시됩니다. SR.get_type() 호출을 사용하여 API를 통해 SR 유형을 쿼리할 수 있습니다.

참고

일부 SR 유형에서는 구성된 블록 크기로 나눌 수 있도록 virtual-size 값을 약간 높게 조정할 수 있습니다.

3.3.1.2. VM에 디스크 이미지 연결

이제 실행 중인 VM(이 예를 시작할 때 있다고 가정했음)과 방금 만든 VDI가 있습니다. 현재 이 둘은 XenServer 호스트에 존재하는 독립적인 개체로, 서로 연결되어 있지 않습니다. 따라서 다음 단계에서는 VDI를 VM과 연결하는 링크를 만듭니다.

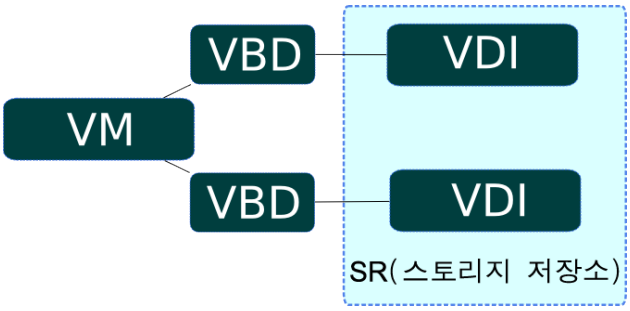
VBD(가상 블록 장치)라는 새 "커넥터" 개체를 만들어 연결을 생성합니다. VBD를 만들기 위해 VBD.create()를 호출합니다. VBD.create() 호출에서는 다음과 같은 여러 매개 변수를 사용합니다.

- VM - VDI를 연결할 VM에 대한 개체 참조입니다.
- VDI - 연결할 VDI에 대한 개체 참조입니다.
- mode - VDI를 읽기 전용으로 연결할지, 읽기-쓰기용으로 연결할지를 지정합니다.
- userdevice - VM 내부에서 실행되는 응용 프로그램이 VDI의 비트를 읽고 쓰는 데 사용할 수 있는 게스트 내부의 블록 장치를 지정합니다.
- type - VDI가 VM 내부에서 일반 디스크로 표시되는지, CD로 표시되는지를 지정합니다. 이러한 특정 필드는 Linux VM보다 Windows VM에서 더 많은 의미를 가집니다. 하지만 이 장에서는 이에 대해 자세히 설명하지 않겠습니다.

VBD.create를 호출하면 XenServer 설치에 VBD 개체가 만들어지며 해당 개체 참조가 반환됩니다. 그러나 이 호출 자체는 실행 중인 VM에서 어떠한 파생 작업도 수행하지 않습니다. 즉, 실행 중인 VM 내부를 살펴보

면 블록 장치가 만들어지지 않은 것을 확인할 수 있습니다. VBD 개체가 존재하지만 게스트에서 블록 장치가 활성화되어 있지 않은 것은 VBD 개체의 currently_attached 필드가 false로 설정되어 있기 때문입니다.

그림 3.2. 두 개의 연결된 VDI가 있는 VM 개체:



두 개의 연결된 VDI가 있는 VM 개체

설명을 위해 그림 3.2. “두 개의 연결된 VDI가 있는 VM 개체”에서는 VM, VBD, VDI 및 SR 간 관계를 표시하는 그래픽 예를 보여 줍니다. 여기서는 VM 개체에 연결된 VDI가 두 개 있고, VM 개체와 해당 VDI 간 연결을 구성하는 VBD 개체가 두 개 있으며, VDI가 같은 SR 내부에 있습니다.

3.3.1.3. VBD 핫플러깅

이 단계에서 VM을 다시 부팅했으면 부팅 후 VBD에 해당하는 블록 장치가 나타났을 것입니다. 부팅 시 XenServer는 VM의 모든 VBD를 쿼리하고 해당하는 VDI를 각각 연결합니다.

VM을 다시 부팅하는 것도 좋지만 여기서 해야 할 작업은 새로 만든 빈 디스크를 실행 중인 VM에 연결하는 것입니다. 이렇게 하려면 새로 만든 VBD 개체에서 plug 메서드를 호출합니다. plug 호출이 성공적으로 반환되면 VBD가 연결된 블록 장치가 실행 중인 VM 내부에 나타납니다. 즉, 실행 중인 VM의 관점에서 보면 새 디스크 장치가 핫 플러그된 것으로 게스트 운영 체제가 인식하게 됩니다. API 관리 환경에서는 이에 따라 VBD의 currently_attached 필드가 true로 설정됩니다.

당연히 VBD 메서드에는 이와 대칭되는 ""가 있습니다. VBD 개체에서 unplug 메서드를 호출하면 연결된 블록 장치가 실행 중인 VM에서 핫 언플러그되고 이에 따라 VBD 개체의 currently_attached 필드가 false로 설정됩니다.

3.3.2. 네트워크 장치를 만들어 VM에 연결

VM의 가상 네트워크 인터페이스 구성과 관련된 API 호출은 여러 가지 점에서 가상 디스크 장치 구성과 관련된 호출과 유사합니다. 따라서 여기서는 API 개체 모델을 사용하여 네트워크 인터페이스를 만드는 방법에 대한 전체 예제를 살펴보지 않고 대신 이 섹션을 사용하여 가상 네트워킹 장치와 가상 스토리지 장치 구성 간에 대칭되는 사항에 대해 간략하게 설명하겠습니다.

네트워킹에서 VBD 클래스에 해당하는 클래스는 VIF 클래스입니다. VBD가 VM 내부 블록 장치의 API 표현인 것처럼 VIF(가상 네트워크 인터페이스)는 VM 내부 네트워크 장치의 API 표현입니다. VBD가 VM 개체를 VDI 개체에 연결하는 반면 VIF는 VM 개체를 네트워크 개체에 연결합니다. VBD와 마찬가지로 VIF에도 VIF와 연결된 게스트 내부의 네트워크 장치가 현재 활성 상태인지 여부를 결정하는 currently_attached 필드가 있습니다. VBD에서 본 것처럼 VM 부팅 시 VM의 VIF가 쿼리되고 각각에 해당하는 네트워크 장치가 부팅되는 VM의 내부에 만들어집니다. 마찬가지로 VIF에도 실행 중인 VM 내/외부에서 네트워크 장치를 핫 플러깅/언플러깅하기 위한 plug 및 unplug 메서드가 있습니다.

3.3.3. 네트워킹 및 스토리지를 위한 호스트 구성

VBD 및 VIF 클래스를 사용하여 VM 내부에서 각각 블록 장치 및 네트워크 장치의 구성을 관리하는 방법에 대해 살펴보았습니다. 스토리지 및 네트워킹 호스트 구성을 관리하는 데 사용되는 두 개의 유사한 클래스인 PBD(Physical Block Device)와 PIF(Physical [network] InterFace)가 있습니다.

3.3.3.1. 호스트 스토리지 구성: PBDs

PBD 클래스부터 살펴보겠습니다. PBD_create() 호출에서는 다음과 같은 여러 매개 변수를 사용합니다.

매개 변수	설명
host	PBD를 사용할 수 있는 물리적 컴퓨터
SR	PBD가 연결되는 스토리지 저장소
device_config	SR이 구현되는 물리적 스토리지 장치를 구성하는 데 필요한 하위 수준 매개 변수를 포함하며, 호스트의 SR 백엔드 드라이버에서 문자열-문자열 맵을 제공합니다. device_config 필드의 구체적인 내용은 PBD가 연결되는 SR 유형에 따라 다릅니다. xe sm-list를 실행하면 가능한 SR 유형 목록이 표시됩니다. 이 열거형의 configuration 필드는 각 SR 유형에 필요한 device_config 매개 변수를 지정합니다.

예를 들어 "nfs" 유형의 SR 개체 s(VDI가 VHD 파일로 저장되어 있는 NFS 파일러의 디렉터리를 나타냄)가 있고 호스트 h가 s에 액세스할 수 있어야 한다고 가정합니다. 이 경우 호스트 h, SR s 및 device_config 매개 변수의 값(아래 맵 참조)을 지정하여 PBD.create()를 호출합니다.

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

이는 SR s가 호스트 h에서 액세스 가능하며 SR s에 액세스하려면 호스트가 디렉터리 /scratch/mysrs/sr1을 my_nfs_server.example.com이라는 NFS 서버에 탑재해야 한다는 것을 XenServer 호스트에 알려 줍니다.

VBD 개체와 마찬가지로 PBD 개체에도 currently_attached라는 필드가 있습니다. 스토리지 저장소는 각각 PBD.plug 및 PBD.unplug 메서드를 호출하여 지정된 호스트에 대해 연결하거나 연결을 끊을 수 있습니다.

3.3.3.2. 호스트 네트워킹 구성: PIF

호스트 네트워크 구성은 PIF 개체를 통해 지정됩니다. PIF 개체가 네트워크 개체 n을 호스트 개체 h에 연결하면 n에 해당하는 네트워크가 PIF 개체의 필드에 지정된 물리적 인터페이스(또는 물리적 인터페이스 + VLAN 태그)에 브리지됩니다.

예를 들어 호스트 h를 네트워크 n에 연결하는 PIF 개체가 있고 이 PIF 개체의 device 필드가 eth0으로 설정되어 있다고 가정합니다. 이는 네트워크 n 상의 모든 패킷이 호스트 네트워크 장치 eth0에 해당하는 호스트의 NIC로 브리지됨을 의미합니다.

3.4. VM 내보내기 및 가져오기

VM을 파일로 내보낸 다음 나중에 임의의 XenServer 호스트로 가져올 수 있습니다. 내보내기 프로토콜은 단순한 HTTP(S) GET이며 이는 VM이 풀 구성원에 있는 경우 마스터에서 수행되어야 합니다. 인증은 표준 HTTP 기본 인증이거나, 세션을 이미 가져온 경우에는 이 세션을 사용할 수 있습니다. 내보내는 VM은 UUID 또는 참조에 의해 지정됩니다. 내보내기를 추적하기 위해 작업을 만든 다음 해당 참조를 사용하여 전달할 수 있습니다. VM의 디스크를 풀 구성원에서만 액세스할 수 있는 경우 요청을 수행하면 리디렉션이 발생할 수 있습니다.

명령줄에서 전달되는 인수는 다음과 같습니다.

인수	설명
session_id	인증에 사용하는 세션에 대한 참조입니다. HTTP 기본 인증을 사용하지 않는 경우에만 필요합니다.

인수	설명
task_id	작업을 추적하는 데 사용하는 작업 개체에 대한 참조입니다. 선택 사항으로, 내보내기를 추적하기 위해 작업 개체를 만든 경우에만 필요합니다.
ref	VM에 대한 참조입니다. UUID를 사용하지 않는 경우에만 필요합니다.
uuid	VM의 UUID입니다. 참조를 사용하지 않는 경우에만 필요합니다.

예를 들어 Linux 명령줄 도구 cURL을 사용하는 경우

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

위 명령은 지정된 VM을 exportfile 파일로 내보냅니다.

메타데이터만 내보내려면 URI http://server/export_metadata를 사용합니다.

가져오기 프로토콜은 이와 유사하게 HTTP(S) PUT를 사용합니다. session_id 및 task_id 인수는 내보내기의 경우와 마찬가지로 사용됩니다. ref 및 uuid는 사용되지 않습니다. 여기서는 VM에 대해 새 참조 및 UUID가 생성됩니다. 여기에는 몇 가지 추가 매개 변수가 있습니다.

인수	설명
restore	true이면 가져오기가 원본 VM을 대체하는 것으로 처리됩니다. 이는 현재 VIF의 MAC 주소가 내보내기의 MAC 주소와 정확하게 일치한다는 것을 의미하며 원본 VM이 아직 실행되고 있는 경우 이로 인해 충돌이 발생할 수 있습니다.
force	true이면 체크섬 실패가 무시됩니다. 기본 동작은 체크섬 오류가 검색될 때 VM을 삭제하는 것입니다.
sr_id	VM을 가져올 SR에 대한 참조입니다. 기본 동작은 Pool.default_SR로 가져오는 것입니다.

예를 들어 cURL을 사용하는 경우

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

위 명령은 VM을 서버의 기본 SR로 가져옵니다.

참고

기본 SR이 설정되어 있지 않고 sr_uuid가 지정되어 있지 않으면 오류 메시지 "DEFAULT_SR_NOT_FOUND"가 반환됩니다.

또 다른 예를 살펴보겠습니다.

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

위 명령은 VM을 서버의 지정된 SR로 가져옵니다.

메타데이터만 가져오려면 URI http://server/import_metadata를 사용합니다.

3.4.1. XVA(Xen Virtual Appliance) VM 가져오기 형식

XenServer에서는 XVA라고 하는 사람이 쉽게 읽을 수 있는 레거시 VM 입력 형식을 지원합니다. 이 섹션에서는 XVA의 구문과 구조에 대해 설명합니다.

XVA는 XML 메타데이터 및 디스크 이미지 집합을 포함하는 디렉터리로 구성됩니다. XVA가 나타내는 VM은 직접 실행할 수 없습니다. XVA 패키지 내 데이터는 압축되어 있고 영구적 스토리지에 보관하거나 XenServer 호스트와 같은 VM 서버로 전송하는 데에만 사용할 수 있습니다. 이 데이터는 VM 서버에서 압축이 해제되고 실행될 수 있습니다.

XVA는 하이퍼바이저 중립적인 패키지 형식이므로 다른 모든 플랫폼에서 XVA VM을 인스턴스화하는 간단한 도구를 만들 수 있습니다. XVA는 특정 런타임 형식을 지정하지 않습니다. 예를 들어 디스크는 파일 이미지, LVM 볼륨, QCoW 이미지, VMDK 또는 VHD 이미지로 인스턴스화될 수 있습니다. XVA VM은 횟수에 제한 없이 인스턴스화될 수 있으며 각 인스턴스는 서로 다른 런타임 형식을 가질 수 있습니다.

XVA는 다음을 수행하지 않습니다.

- 특정 일련화 또는 전송 형식 지정
- 설치 시 VM(또는 템플릿)을 사용자 지정하는 메커니즘 제공
- 설치 후 VM이 업그레이드되는 방법 제시
- 장비 역할을 수행하는 여러 VM이 통신하는 방법 정의

이러한 항목은 모두 관련 Open Virtual Appliance 사양에서 처리됩니다.

XVA는 최소한 ova.xml이라는 파일을 포함하는 디렉터리입니다. 이 파일은 XVA에 포함된 VM을 정의하며 섹션 3.2에서 설명합니다. 디스크는 하위 디렉터리에 저장되며 ova.xml에서 참조됩니다. 디스크 데이터 형식은 나중에 섹션 3.3에서 설명합니다.

이 장의 나머지 부분에서는 다음 용어를 사용합니다.

- HVM: 수정되지 않은 OS 커널이 하드웨어에서 가상화 지원의 도움으로 실행되는 모드
- PV: 특수하게 수정된 "반가상화" 커널이 가상화를 위한 하드웨어 지원을 필요로 하지 않고 하이퍼바이저를 기반으로 명시적으로 실행되는 모드

"ova.xml" 파일에는 다음 요소가 포함되어 있습니다.

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

"version" 특성의 숫자는 XVA가 생성된 사양의 버전을 나타내며 여기서는 버전 0.1입니다. <appliance> 내에는 <vm>이 단 한 개뿐이지만 OVA 사양에서는 여러 개의 <vm>이 허용됩니다.

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

각 <vm> 요소는 하나의 VM을 정의합니다. "name" 특성은 향후 내부에서 사용하기 위한 것이며 ova.xml 파일 내에서 고유해야 합니다. "name" 특성은 임의의 유효한 UTF-8 문자열이 될 수 있습니다. 각 <vm> 태그 내부에는 다음과 같은 필수 요소가 있습니다.

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

UI에 표시되는 VM의 약식 이름입니다.

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

UI에 표시되는 VM에 대한 설명입니다. <label> 및 <shortdesc>의 내용에서는 선행 및 후행 공백이 무시됩니다.

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

<config> 요소에는 VM이 보유해야 하는 바이트 단위 메모리 크기(mem_set) 및 CPU 수(VCPU)를 정의하는 특성이 있습니다.

각 <vm>에는 다음과 같이 블록 장치를 나타내는 <vbd> 요소가 0개 이상 있습니다.

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

각 특성의 의미는 다음과 같습니다.

device	VM에 표시할 물리적 장치 이름입니다. linux 게스트인 경우 "sd[a-z]"를 사용하고 windows 게스트인 경우 "hd[a-d]"를 사용합니다.
function	"root"로 표시된 경우 이 디스크는 게스트를 부팅하는 데 사용됩니다(이것이 Linux 루트, 즉 / filesystem이 있음을 의미하지는 않음). 하나의 장치만 "root"로 표시될 수 있습니다. VM 부팅에 대한 설명은 섹션 3.4를 참조하십시오. 다른 모든 문자열은 무시됩니다.
mode	장치가 읽기/쓰기이거나 읽기 전용인 경우 "w" 또는 "ro"입니다.
vdi	이 블록 장치가 연결된 디스크 이미지의 이름(<vdi> 요소로 나타남)입니다.

각각의 <vm>에는 <hacks is_hvm="false" kernel_boot_cmdline="root=/dev/sda1 ro"/>와 같은 선택적인 <hacks> 섹션이 포함될 수 있습니다. <hacks> 요소는 XenServer에서 생성하는 XVA 파일에 있지만 향후 제거될 예정입니다. "is_hvm" 특성은 VM을 HVM에서 부팅해야 하는지 여부에 따라 "true" 또는 "false" 중 하나입니다. "kernel_boot_cmdline"에는 pygrub을 사용하여 게스트를 부팅할 때 사용되는 추가 커널 명령줄 인수가 포함됩니다.

<vm> 요소 외에도 <appliance>에는 다음과 같은 <vdi> 요소가 0개 이상 포함됩니다.

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

각 <vdi>는 디스크 이미지에 해당합니다. 각 특성의 의미는 다음과 같습니다.

- name: <vbd> 요소의 vdi 특성이 참조하는 VDI의 이름입니다. 임의의 유효한 UTF-8 문자열이 허용됩니다.
- size: 필요한 이미지의 크기(바이트)입니다.
- source: 이미지에 해당하는 데이터의 위치를 설명하는 URI입니다. 현재는 file:// URI만 사용할 수 있으며, 이 URI는 ova.xml이 들어 있는 디렉터리에 상대적인 경로를 설명해야 합니다.
- type: 디스크 데이터의 형식을 정의합니다(섹션 3.3. 참조).

"dir-gzipped-chunks" 형식의 단일 디스크 이미지 인코딩을 지정합니다. 각각의 이미지는 다음과 같은 파일 시퀀스를 포함한 디렉터리로 표시됩니다.

```
-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
```

이름이 "chunk-XXXXXXXXX.gz"인 각 파일은 정확하게 1e9 바이트(1GiB가 아닌 1GB)의 원시 블록 데이터를 포함하는 gzip된 파일입니다. 여러 파일 시스템의 최대 파일 크기 제한 미만으로 안전하게 유지되도록 작은 크기가 선택되었습니다. 파일을 gunzip한 다음 서로 연결하면 원래 이미지가 복구됩니다.

XenServer에서는 다음과 같이 두 가지 VM 부팅 메커니즘을 제공합니다. (1) pygrub을 통해 추출된 반가상화 커널 사용 (2) HVM 사용 현재 구현에서는 <hacks> 섹션의 "is_hvm" 플래그를 통해 사용할 메커니즘을 결정합니다.

이 섹션의 나머지에서는 XVA로 패키지된 매우 간단한 Debian VM에 대해 설명합니다. 이 VM에는 크기가 5120MiB이고 루트 파일 시스템과 pygrub을 통한 게스트 부팅에 사용되는 디스크 하나와, 크기가 512MiB이고 스왑에 사용되는 디스크 하나, 이렇게 두 개의 디스크가 있습니다. VM에는 512MiB의 메모리가 있으며 이 VM은 하나의 가상 CPU를 사용합니다.

최상위 수준에서 간단한 Debian VM은 단일 디렉터리로 표시됩니다.

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

주 XVA 디렉터리 내에는 디스크당 하나씩 두 개의 하위 디렉터리와 ova.xml이라는 하나의 파일이 있습니다.

```
$ 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
```

각 디스크 하위 디렉터리 내에는 일련의 파일이 있는데 각 파일에는 gzip을 사용하여 압축된 1GB의 원시 디스크 블록 데이터가 포함됩니다.

```
$ 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
```

간단한 Debian VM 예에는 다음과 같은 XVA 파일이 있을 수 있습니다.

```
<?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 참고 사항

3.5.1. 날짜 및 시간

API는 날짜 및 시간 처리와 관련하여 XML-RPC 사양을 그대로 따르지 않습니다. API는 날짜 및 시간 문자열의 끝에 "Z"를 추가하여 시간이 UTC로 표현됨을 나타냅니다.

3.6. 기타 참조 자료

이 장에서는 API와 해당 개체 모델에 대해 개략적으로 살펴보았습니다. 여기서의 목표는 API의 자세한 의미 체계를 보여 주는 것이 아니라 다음 장의 코드 샘플을 읽을 수 있고 더 자세한 XenServer API 참조 문서를 참고하여 작업을 진행할 수 있을 정도의 배경 지식을 제공하는 것입니다.

다음 자료에서 더 자세한 정보를 찾아볼 수 있습니다.

- XenServer 관리자 가이드에는 xe CLI에 대한 개요가 포함되어 있습니다. xe 명령의 많은 부분이 API와 거의 유사하므로 xe를 살펴보면 이 장에서 설명하는 API 개체 모델을 이해하는 데 도움이 됩니다.
- 다음 장의 코드 샘플에서는 다양한 클라이언트 언어로 작성된 몇 가지 구체적인 API 코딩 인스턴스를 제공합니다.
- XenServer API 참조 문서에서는 API 의미 체계를 더 자세히 설명하고 네트워크상의 XML/RPC 메시지 형식에 대해 설명합니다.
- XenServer 호스트 dom0 자체에도 API를 사용하는 몇 개의 스크립트가 있습니다. 예를 들어 "/opt/xensource/libexec/shutdown"은 VM을 완전하게 종료하는 python 프로그램입니다. 이 스크립트는 호스트 자체가 종료될 때 호출됩니다.

CITRIX

4장. API 사용

이 장에서는 실제 프로그램에서 XenServer Management API를 사용하여 XenServer 호스트 및 VM을 관리하는 방법에 대해 설명합니다. 이 장에서는 먼저 일반적인 클라이언트 응용 프로그램에 대해 살펴본 후 API를 사용하여 일반적인 작업을 수행하는 방법에 대해 알아봅니다. 코드 조각 예는 python 구문으로 작성되어 있지만 같은 코드를 다른 프로그래밍 언어로 작성해도 이와 비슷합니다. 이 장의 마지막은 완전한 예제 2개로 구성되어 있습니다.

4.1. 일반적인 응용 프로그램 분석

이 섹션에서는 XenServer Management API를 사용하는 일반적인 응용 프로그램 구조에 대해 설명합니다. 대부분의 클라이언트 응용 프로그램은 먼저 XenServer 호스트에 연결하고 사용자 이름, 암호 등을 사용하여 인증을 수행합니다. 인증이 성공하는 경우 서버는 "세션" 개체를 만들고 클라이언트에 참조를 반환합니다. 이 참조는 향후 모든 API 호출에 인수로 전달됩니다. 인증이 완료되면 클라이언트가 XenServer 호스트, VM 등과 같은 다른 유용한 개체에 대한 참조를 검색하고 해당 개체에 대한 작업을 호출할 수 있습니다. 작업은 동기 또는 비동기로 호출될 수 있으며 특수 작업 개체가 비동기 작업의 상태 및 진행률을 나타냅니다. 이러한 응용 프로그램 요소는 모두 다음 섹션에 자세하게 설명합니다.

4.1.1. 하위 수준 전송 선택

API 호출은 두 가지 전송을 통해 수행할 수 있습니다.

- IP 네트워크를 통해 포트 443(https)을 사용하는 SSL로 암호화된 TCP 전송
- 로컬 Unix 도메인 소켓을 통한 일반 텍스트 전송: /var/xapi/xapi

SSL로 암호화된 TCP 전송은 모든 오프 호스트 트래픽에 사용되고, Unix 도메인 소켓은 XenServer 호스트 자체에서 직접 실행되는 서비스에서 사용될 수 있습니다. SSL로 암호화된 TCP 전송에서 모든 API 호출은 리소스 풀 마스터로 연결되어야 합니다. 이렇게 하지 못할 경우 오류 매개 변수로 마스터의 IP 주소를 포함하는 HOST_IS_SLAVE 오류가 발생합니다.

풀의 마스터 호스트는 변경될 수 있으므로(특히 풀에서 HA를 사용하는 경우) 클라이언트는 마스터 호스트 변경을 감지하고 필요에 따라 새 마스터에 연결할 수 있도록 다음과 같은 단계를 구현해야 합니다.

풀 마스터 변경 처리:

1. 호스트 서버 목록의 업데이트를 구독하고 풀의 현재 호스트 목록을 유지 관리합니다.
2. 풀 마스터에 대한 연결에서 응답에 실패한 경우 응답하는 호스트가 있을 때까지 목록의 모든 호스트에 연결을 시도합니다.
3. 응답하는 첫 번째 호스트는 새 마스터가 아닌 경우 새 풀 마스터의 ID가 포함된 HOST_IS_SLAVE 오류 메시지를 반환합니다.
4. 새 마스터에 연결합니다.

참고

특수한 경우로 Unix 도메인 소켓을 통해 전송되는 모든 메시지는 올바른 노드로 투명하게 전달됩니다.

4.1.2. 인증 및 세션 처리

API 호출의 대부분은 세션 참조를 첫 번째 매개 변수로 받습니다. 올바른 참조를 제공하지 못하면 SESSION_INVALID 오류가 반환됩니다. 함수에 사용자 이름 및 암호를 제공하여 세션 참조를 얻습니다.

참고

특수한 경우로 이 호출이 로컬 Unix 도메인 소켓을 통해 실행되면 사용자 이름 및 암호가 무시되고 호출이 항상 성공합니다.

모든 세션에는 API를 호출할 때마다 업데이트되는 "last active"라는 타임스탬프가 연결되어 있습니다. 서버 소프트웨어에서는 현재 기본 제공 활성 세션 수의 제한이 500개이며 지정된 username 또는 originator에 대해 이 제한을 초과하면 "last active" 필드가 가장 오래된 세션부터 제거됩니다. 또한 "last active" 필드가 24시간보다 오래된 모든 세션도 제거됩니다. 따라서 다음을 수행하는 것이 중요합니다.

- 로그인 시 적절한 originator를 지정하고
- 세션이 누출되지 않도록 활성 세션에서 로그아웃합니다.
- SESSION_INVALID 오류가 발생하는 경우 서버에 다시 로그인할 수 있도록 준비합니다.

다음 Python 조각에서는 Unix 도메인 소켓을 통해 연결이 설정되고 세션이 만들어집니다.

```
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. 유용한 개체에 대한 참조 찾기

응용 프로그램이 인증된 후 다음 단계는 개체의 상태를 쿼리하고 해당 개체에 대한 작업을 호출할 수 있도록 개체에 대한 참조를 얻는 것입니다. 모든 개체는 다음 내용을 포함하는 일련의 "암시적인" 메시지를 가지고 있습니다.

- get_by_name_label: 특정 레이블을 가진 특정 클래스의 모든 개체 목록을 반환합니다.
- get_by_uuid: 해당 UUID로 이름이 지정된 단일 개체를 반환합니다.
- get_all: 특정 클래스의 모든 개체에 대한 참조 집합을 반환합니다.
- get_all_records: 특정 클래스의 각 개체의 레코드에 대한 참조 맵을 반환합니다.

예를 들어 모든 호스트를 나열하려면 다음을 실행합니다.

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

이름이 "my first VM"인 모든 VM을 찾으려면 다음을 실행합니다.

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

참고

개체의 name_label 필드는 고유하지 않을 수 있으므로 get_by_name_label API 호출은 단일 참조보다는 참조 집합을 반환합니다.

위에서 설명한 개체 찾기 방법 외에도 대부분의 개체에는 필드 내에 다른 개체에 대한 참조도 들어 있습니다. 예를 들어 다음을 호출하여 특정 호스트에서 실행 중인 VM 집합을 찾을 수 있습니다.

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

4.1.4. 개체에 대한 동기 작업 호출

개체 참조를 얻었으면 해당 개체에 대한 작업을 호출할 수 있습니다. 예를 들어 VM을 시작하려면 다음을 실행합니다.

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

모든 API 호출은 기본적으로 동기 호출이며 작업이 완료되거나 실패할 때까지 반환되지 않습니다. 예를 들어 VM.start의 경우 VM 부팅이 시작될 때까지 호출이 반환되지 않습니다.

참고

VM.start 호출이 반환되면 VM이 부팅됩니다. 부팅이 언제 완료되었는지 파악하려면 게스트 내부 에이전트가 VM_guest_metrics 개체를 통해 내부 통계를 보고할 때까지 대기합니다.

4.1.5. 작업을 사용하여 비동기 작업 관리

상당히 오랜 시간이 걸리는 작업(예: 및)의 관리를 간소화하기 위해 동기(기본값) 및 비동기라는 두 가지 형태로 함수가 제공됩니다. 각 비동기 함수는 다음을 포함하여 진행 중인 작업에 대한 정보가 포함된 작업 개체에 대한 참조를 반환합니다.

- 보류 중인지 여부
- 성공 또는 실패 여부
- 진행률(0-1 범위)
- 작업에서 반환된 결과 또는 오류 코드

VM.clone 작업의 진행률을 추적하고 진행률 표시줄을 표시하려는 응용 프로그램에서는 다음과 같은 코드를 사용할 수 있습니다.

```
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)
```

참고

올바르게 동작하는 클라이언트에서는 결과 또는 오류를 읽은 후 비동기 작업에서 생성한 작업을 삭제해야 합니다. 작업 수가 기본 제공 임계값을 초과하는 경우 서버에서는 완료된 작업 중 가장 오래된 작업을 삭제합니다.

4.1.6. 이벤트 구독 및 수신

작업 및 메트릭 클래스를 제외하고 개체가 수정될 때마다 서버에서 이벤트를 생성합니다. 클라이언트에서는 자주 폴링하는 대신 클래스별로 이러한 이벤트 스트림을 구독하여 업데이트를 수신할 수 있습니다. 이벤트의 유형에는 다음 세 가지가 있습니다.

- add - 개체가 만들어지면 생성됩니다.

- del - 개체가 삭제되기 직전에 생성됩니다.
- mod - 개체 필드가 변경되면 생성됩니다.

이벤트에는 일정하게 증가하는 ID, 개체 클래스의 이름 및 get_record() 결과와 동일한 개체 상태의 스냅샷도 포함되어 있습니다.

클라이언트는 클래스 이름 목록 또는 특수 문자열 "*"과 함께 event.register()를 호출하여 이벤트에 등록합니다. 클라이언트는 이벤트를 사용할 수 있을 때까지 차단하고 새 이벤트를 반환하는 event.next()를 실행하여 이벤트를 수신합니다.

참고

서버에 생성된 이벤트 대기열의 길이는 제한되어 있으므로 매우 느린 클라이언트는 이벤트를 다 읽지 못할 수 있습니다. 이런 경우 EVENTS_LOST 오류가 반환됩니다. 클라이언트는 이벤트에 다시 등록하고 등록이 해제된 동안 대기 중인 조건이 실현되지 않은 것을 확인하여 이러한 상황을 처리할 준비를 해야 합니다.

다음 Python 코드 조각에서는 시스템에서 생성된 모든 이벤트 요약을 출력하는 방법을 보여 줍니다. 이와 유사한 코드는 Xenserver-SDK/XenServerPython/samples/watch-all-events.py에 포함되어 있습니다.

```
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. 전체 응용 프로그램 예제

이 섹션에서는 API를 사용하는 실제 프로그램의 전체 예제 두 개에 대해 설명합니다.

4.2.1. XenMotion을 사용하여 동시 VM 마이그레이션

XenServer-SDK/XenServerPython/samples/permute.py에 포함되어 있는 이 python 예제는 XenMotion을 사용하여 리소스 풀에서 호스트 간 VM을 동시에 이동하는 방법에 대해 보여 줍니다. 이 예제에서는 비동기 API 호출을 사용하고 일련의 작업이 완료되기까지 대기하는 방법을 보여 줍니다.

이 프로그램은 몇 가지 표준 상용구로 시작하여 API 바인딩 모듈을 가져옵니다.

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

다음에는 서버 URL, 사용자 이름, 암호 및 반복 수가 포함된 명령줄 인수가 구문 분석됩니다. 사용자 이름 및 암호는 루프에서 여러 번 호출되는 main 함수에 전달되는 세션을 설정하는 데 사용됩니다. 끝 부분에서 프로그램이 세션을 로그아웃하도록 try: finally:가 사용된 것을 확인할 수 있습니다.

```
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 함수는 시스템에서 실행 중인 각 VM을 검사하고 제어 도메인(시스템의 일부로 사용자가 제어할 수 없음)을 필터링하여 제외합니다. 실행 중인 VM 및 해당 현재 호스트 목록이 생성됩니다.

```
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))
```

다음에는 호스트 목록이 회전됩니다.

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

이러한 회전에서 각 VM은 XenMotion을 사용하여 새 호스트로 이동됩니다. 예를 들어 목록의 위치 2에 있는 호스트에서 실행 중인 VM은 목록의 위치 1에 있는 호스트로 이동됩니다. 각 이동을 병렬로 실행하기 위해 VM.pool_migrate의 비동기 버전이 사용되고 작업 참조 목록이 생성됩니다. live 플래그가 VM.pool_migrate에 전달되는 것을 알 수 있습니다. 이렇게 하면 VM이 계속 실행 중인 경우 이동됩니다.

```
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)
```

이제 완료 여부를 확인하기 위해 작업 목록이 폴링됩니다.

```
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)
```

모든 작업이 pending 상태를 벗어나면(즉, 성공적으로 완료되었거나 실패했거나 취소됨) 작업이 모두 성공했는지 확인하기 위해 다시 한 번 폴링됩니다.

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

실패한 작업이 있는 경우 세부 정보가 출력되고, 예외가 발생하며, 작업 개체가 추가 검사를 위해 그대로 유지됩니다. 모든 작업이 성공하면 작업 개체가 삭제되고 함수가 반환됩니다.

```
if not(allok):
      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)
```

4.2.2. XE CLI를 사용하여 VM 복제

이 예제는 XE CLI를 사용하여 VM을 복제하는 bash 스크립트입니다. 여기서는 VM의 전원이 켜진 경우 이를 먼저 종료하는 작업도 처리합니다.

이 예제는 먼저 환경 변수 XE가 설정되어 있는지 확인하는 몇 가지 상용구로 시작합니다. 이 환경 변수가 설정되어 있으면 해당 변수가 CLI의 전체 경로를 가리키고 있다고 가정하고 그렇지 않은 경우 XE CLI가 현재 경로에 있다고 가정합니다. 다음으로 이 스크립트에서는 사용자에게 서버 이름, 사용자 이름 및 암호를 입력하라는 프롬프트를 표시합니다.

```
# 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
```

그런 다음 스크립트에서는 해당 명령줄 인수를 확인합니다. 여기에는 복제되는 VM의 UUID인 인수만 필요합니다.

```
# 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
```

그런 다음 이 스크립트는 VM의 전원 상태를 확인하고 실행 중인 경우 완전한 종료를 시도합니다. VM이 "Halted"(중지됨) 상태로 전환될 때까지 대기하는 데 이벤트 시스템이 사용됩니다.

참고

XE CLI에서는 추가 공백이나 형식 지정 없이 해당 출력을 출력하도록 하는 명령줄 인수 --minimal(스크립트에서 사용하기에 적합함)을 지원합니다. 값이 여러 개 반환되는 경우 값은 쉼표로 구분됩니다.

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

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

그러면 VM이 복제되고 새 VM의 name_label이 cloned_vm으로 설정됩니다.

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

마지막으로 원본 VM이 실행 중이었는데 종료되었으면 해당 VM과 새 VM이 시작됩니다.

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

CITRIX

5장. HTTP를 사용하여 XenServer와 상호 작용

XenServer에서는 각 호스트에서 HTTP 인터페이스를 표시합니다. 이 인터페이스를 사용하여 다양한 작업을 수행할 수 있습니다. 이 장에서는 사용할 수 있는 메커니즘에 대해 설명합니다.

5.1. VM 가져오기 및 내보내기

VM을 가져오고 내보내는 데 시간이 다소 걸릴 수 있으므로 가져오기 및 내보내기 작업에 대한 비동기 HTTP 인터페이스가 제공됩니다. XenServer API를 사용하여 내보내기를 수행하려면 다음 의사 코드에 표시되어 있는 것과 같이 유효한 세션 ID, 작업 ID 및 VM UUID를 지정하여 HTTP GET 호출을 생성해야 합니다.

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

가져오기 작업의 경우 다음 의사 코드에 나와 있는 것과 같이 HTTP PUT 호출을 사용합니다.

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

5.2. XenServer 성능 통계 가져오기

XenServer에서는 XenServer 설치의 다양한 부분의 성능에 대한 통계를 기록합니다. 메트릭은 장기적인 액세스와 시간에 따른 추세 분석을 위해 영구적으로 저장됩니다. VM에서 스토리지를 사용할 수 있는 경우 VM이 종료될 때 통계가 디스크에 기록됩니다. 통계는 RRD(라운드 로빈 데이터베이스)에 저장되는데 RRD는 개별 VM(제어 도메인 포함) 및 서버에 대해 유지 관리됩니다. RRD는 VM이 실행되고 있는 서버 또는 VM이 실행되지 않는 경우 풀 마스터에 저장됩니다. 또한 RRD는 매일 백업됩니다.

주의

이전 버전의 XenServer API에서는 VM_metrics, VM_guest_metrics, host_metrics 메서드와 관련 메서드를 사용하여 즉각적인 성능 메트릭을 얻을 수 있었습니다. 이러한 메서드는 이제 사용되지 않으며 대신 이 장에서 설명하는 http 처리기를 사용하여 VM 및 서버의 RRD에서 통계를 다운로드합니다. 기본적으로 레거시 메트릭은 0을 반환합니다. 이전 버전의 XenServer에서 사용했던 주기적 통계 폴링 상태로 돌아가려면 호스트에서 other-config:rrd_update_interval=<interval> 매개 변수를 다음 값 중 하나로 설정하고 호스트를 다시 시작합니다.

never	기본값이며 주기적 폴링을 수행하지 않음을 의미합니다.
1	폴링이 5초마다 수행됩니다.
2	폴링이 1분마다 수행됩니다.

기본적으로 이전 메트릭 API는 값을 반환하지 않으므로 레거시 모니터링 프로토콜을 사용하는 모니터링 클라이언트를 실행하려면 이 키를 활성화해야 합니다.

통계는 최대 1년간 유지되며 다양하게 세분화되어 저장됩니다. 평균 및 최신 값은 다음 간격으로 저장됩니다.

- 지난 10분: 5초
- 지난 2시간: 1분
- 지난 주: 1시간

- 지난 연도: 1일

RRD는 디스크에 압축되지 않은 XML로 저장됩니다. 디스크에 기록되는 각 RRD의 크기는 200KiB에서 약 1.2MiB(RRD가 1년 동안의 통계를 저장하는 경우) 사이입니다.

주의

디스크에 빈 공간이 없는 등의 이유로 통계를 디스크에 쓸 수 없는 경우 통계는 손실되고 마지막으로 저장된 RRD 버전이 사용됩니다.

통계는 wget 등을 사용하여 XML 형식으로 HTTP를 통해 다운로드할 수 있습니다. XML 형식에 대한 자세한 내용은 http://oss.oetiker.ch/rrdtool/doc/rrddump.en.html 및 http://oss.oetiker.ch/rrdtool/doc/rrdxport.en.html을 참조하십시오. HTTP 인증에서는 사용자 이름 및 암호 또는 세션 토큰을 사용할 수 있습니다. 매개 변수는 URL 다음 물음표(?) 뒤에 추가되며 앰퍼샌드(&)로 구분됩니다.

호스트에서 모든 VM 통계의 업데이트를 가져오려는 경우 URL의 형식은 다음과 같습니다.

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

이 요청은 쿼리되는 특정 호스트에 있는 모든 VM에 대해 rrdtool xport 스타일의 XML 형식으로 데이터를 반환합니다. 내보내기에서 어떤 열이 어떤 VM과 관련되는지 구분하기 위해 legend 필드 앞에 VM의 UUID가 추가됩니다.

호스트 업데이트도 가져오려면 쿼리 매개 변수 host=true를 사용합니다.

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

기간이 줄어들면 단계 역시 줄어듭니다. 즉, 더 짧은 기간에 대한 통계를 요청하면 통계가 더 자세히 반환됩니다.

추가 rrd_updates 매개 변수

cf= <ave|min|max>	데이터 통합 모드
interval= <interval>	보고할 값 사이의 간격

참고

기본적으로 ave 통계만 사용할 수 있습니다. VM에 대한 min 및 max 통계를 얻으려면 다음 명령을 실행합니다.

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

호스트에 대한 모든 통계를 얻으려면 다음을 수행합니다.

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

VM에 대한 모든 통계를 얻으려면 다음을 수행합니다.

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

CITRIX

6장. XenServer API 확장

XenAPI는 가상 컴퓨터의 수명주기를 관리하는 일반적이고 포괄적인 인터페이스로, XenAPI를 사용하여 XenAPI 공급자는 특정 기능(예: 스토리지 프로비저닝 또는 콘솔 처리)을 유연하게 구현할 수 있습니다. XenServer에는 자체 XenCenter 인터페이스에서 사용되는 유용한 기능을 제공하는 몇 가지 확장이 있습니다. 이 장에서는 이러한 메커니즘의 작동 방식에 대해 설명합니다.

XenAPI에 대한 확장은 다양한 개체에 대한 other-config 맵 키를 지정하여 제공되는 경우가 많습니다. 이 매개 변수의 사용은 기능이 특정 XenServer 릴리스에 대해 지원되지만 장기적인 기능은 아님을 나타냅니다. API로의 기능 승격에 대해 지속적으로 평가하고 있지만 이렇게 하려면 인터페이스의 특성을 잘 이해해야 합니다. 이러한 확장을 어떻게 사용하고 있는지에 대한 개발자 피드백은 이러한 결정을 내리는 데 도움이 되므로 언제든지 보내 주십시오.

6.1. VM 콘솔 전달

대부분의 XenAPI 그래픽 인터페이스는 사용자에게 물리적 컴퓨터인 것처럼 보이기 위해 VM 콘솔에 액세스하려고 합니다. 게스트 유형이나 물리적 호스트 콘솔에 액세스 중인지 여부에 따라 사용할 수 있는 몇 가지 콘솔 유형이 있습니다.

콘솔 액세스

운영 체제	텍스트	그래픽	최적화된 그래픽
Windows	아니요	VNC, API 호출을 통해	RDP, 게스트에서 직접
Linux	예, VNC 및 API 호출을 통해	아니요	VNC, 게스트에서 직접
물리적 호스트	예, VNC 및 API 호출을 통해	아니요	아니요

Windows와 같이 하드웨어에서 지원하는 VM은 VNC를 통해 직접 그래픽 콘솔을 제공합니다. 텍스트 기반 콘솔은 없으며 그래픽 콘솔을 사용하는 데 게스트 네트워킹은 필요하지 않습니다. 게스트 네트워킹이 설정된 경우에는 원격 데스크톱 액세스를 설치하고 RDP 클라이언트를 사용하여 직접 연결하는 것이 더 효율적입니다(XenAPI 외부에서 수행됨).

Linux 게스트와 같은 반가상화 VM은 기본 텍스트 콘솔을 직접 제공합니다. XenServer는 이러한 텍스트 기반 콘솔을 그래픽 VNC 표현으로 변환하는 vncterm이라는 유틸리티를 제공합니다. 이 콘솔이 기능하는 데 게스트 네트워킹은 필요하지 않습니다. 위 Windows의 경우와 마찬가지로 Linux 배포판에서는 게스트 내부에 VNC를 구성하고 게스트 네트워크 인터페이스를 통해 직접 연결하는 경우가 많습니다.

물리적 호스트 콘솔은 vt100 콘솔로만 사용할 수 있으며 제어 도메인에서 vncterm을 사용하여 XenAPI를 통해 VNC 콘솔로 표시됩니다.

RFB(Remote Framebuffer)는 VNC의 기반이 되는 프로토콜로 The RFB Protocol에 명시되어 있습니다. 타사 개발자는 자체 VNC 뷰어를 제공해야 하는데 이를 위해 무료로 사용할 수 있는 여러 가지 구현을 변경하여 사용할 수 있습니다. 뷰어에서 지원해야 하는 최소 버전은 RFB 3.3입니다.

6.1.1. API를 사용하여 VNC 콘솔 검색

VNC 콘솔은 호스트 에이전트에 전달되는 특수 URL을 사용하여 검색됩니다. API 호출 순서는 다음과 같습니다.

1. 클라이언트에서 Master/443으로: XML-RPC: Session.login_with_password().
2. Master/443에서 클라이언트로: 후속 호출에 사용될 세션 참조를 반환합니다.
3. 클라이언트에서 Master/443으로: XML-RPC: VM.get_by_name_label().
4. Master/443에서 클라이언트로: 특정 VM(또는 물리적 호스트 콘솔을 검색하려는 경우 "제어 도메인")에 대한 참조를 반환합니다.
5. 클라이언트에서 Master/443으로: XML-RPC: VM.get_consoles().
6. Master/443에서 클라이언트로: VM과 연결된 콘솔 개체 목록을 반환합니다.
7. 클라이언트에서 Master/443으로: XML-RPC: VM.get_location().
8. 요청된 콘솔이 있는 위치를 설명하는 URI를 반환합니다. URI는 다음과 같은 형식을 사용합니다. https://192.168.0.1/console?ref=OpaqueRef:c038533a-af99-a0ff-9095-c1159f2dc6a0
9. 클라이언트에서 192.168.0.1로: HTTP CONNECT "/console?ref=(...)"

HTTP/1.1 RFC에서는 HTTP CONNECT가 URL이 아니라 호스트와 포트여야 한다고 지정하기 때문에 마지막 HTTP CONNECT는 조금 표준에서 벗어나 있습니다. HTTP 연결이 완료되면 그 이후로는 추가적인 HTTP 프로토콜 작업 없이 이 연결을 직접 VNC 서버로 사용할 수 있습니다.

이 스키마에서는 클라이언트에서 제어 도메인의 IP에 직접 액세스해야 하며 이러한 연결을 차단하는 NAT(Network Address Translation) 장치가 있는 경우에는 제대로 작동하지 않습니다. CLI를 사용하여 클라이언트에서 콘솔 URI를 검색하여 연결을 확인할 수 있습니다.

CLI를 사용하여 콘솔 URI를 검색하려면:

1. 다음을 실행하여 VM UUID를 검색합니다.

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

2. 콘솔 정보를 검색합니다.

```
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 같은 명령줄 유틸리티를 사용하여 location 필드에 지정된 IP 주소에 대한 연결을 테스트합니다.

6.1.2. Linux VM에 대한 VNC 전달 비활성화

Linux VM을 만들고 삭제할 때 호스트 에이전트는 텍스트 콘솔을 VNC로 변환하는 vncterm 프로세스를 자동으로 관리합니다. 텍스트 콘솔에 직접 액세스하려는 고급 사용자는 해당 VM에 대한 VNC 전달을 비활성화할 수 있습니다. 그러면 텍스트 콘솔은 제어 도메인을 통해서만 직접 액세스할 수 있으며 XenCenter와 같은 그래픽 인터페이스는 해당 VM에 대해 콘솔을 렌더링할 수 없습니다.

CLI를 사용하여 Linux VNC 콘솔 비활성화:

1. 게스트를 시작하기 전에 VM 레코드에서 다음 매개 변수를 설정합니다.

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

2. VM을 시작합니다.
3. 다음과 같이 CLI를 사용하여 VM의 기본 도메인 ID를 검색합니다.

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

4. 호스트 콘솔에서 다음을 실행하여 직접 텍스트 콘솔에 연결합니다.

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

이 구성은 고급 절차로 I/O 작업이 많은 경우에는 텍스트 콘솔을 직접 사용하지 않는 것이 좋습니다. 대신 SSH 또는 일부 다른 네트워크 기반 연결 메커니즘을 통해 게스트에 연결하십시오.

6.2. 반가상화 Linux 설치

반가상화 Linux 게스트 설치는 하드웨어 지원을 통해 단순히 게스트를 설치하면 되는 것이 아니라 Xen을 인식하는 커널을 부팅해야 하므로 조금 복잡합니다. 이렇게 하면 에뮬레이션 오버헤드가 없으므로 기본 설치 속도에 근접하게 되는 장점이 있습니다. XenServer는 여러 다른 Linux 배포 설치를 지원하며 이 과정을 최대한 추상화합니다.

이를 위해 eliloader라는 특별한 bootloader가 제어 도메인에 존재하며 이 bootloader는 시작 시 VM 레코드의 다양한 other-config 키를 읽고 배포별 설치 동작을 수행합니다.

- install-repository - 필수 사항. 저장소에 대한 경로입니다('http', 'https', 'ftp', 또는 'nfs'). 대상 설치 관리자에서 사용되도록 지정해야 하지만 접두사(예: method=)를 포함하지 않아야 합니다.
- install-vnc - 기본값: false. 설치 중 사용 가능한 경우 VNC를 사용합니다.
- install-vncpasswd - 기본값: 비어 있음. 대상 배포의 명령줄을 사용하여 제공할 수 있는 경우 사용할 VNC 암호입니다.
- install-round - 기본값: 1. 현재 bootloader 라운드입니다. 사용자가 편집할 수 없습니다(아래 참조).

6.2.1. Red Hat Enterprise Linux 4.1/4.4

eliloader는 부팅의 두 라운드에 사용됩니다. 첫 번째 라운드에서는 설치 관리자 initrd 및 커널을 /opt/xensource/packages/files/guest-installer에서 반환합니다. 그런 다음 두 번째 부팅에서는 VM에서 추가 업데이트 디스크를 제거하고, bootloader를 pygrub으로 전환한 후 정상적인 부팅을 시작합니다.

이러한 시퀀스가 필요한 이유는 Red Hat에서 이러한 배포에 대해 Xen 커널을 제공하지 않고 따라서 이러한 배포에 대해 XenServer 사용자 지정 커널이 대신 사용되기 때문입니다.

6.2.2. Red Hat Enterprise Linux 4.5/5.0

커널 및 ramdisk가 사용자가 지정한 네트워크 저장소에서 직접 다운로드되고 bootloader를 직접 pygrub으로 전환하는 점을 제외하면 RHEL4.4 설치와 유사합니다. pygrub은 직접 실행되지 않으므로 다음 부팅 시에만 구문 분석됩니다.

네트워크 검색을 통해 사용자는 사용자의 네트워크 저장소에서 직접 업스트림 Red Hat 공급업체 커널을 설치할 수 있습니다. 또한 업데이트된 XenServer 커널이 여러 Xen 관련 버그를 수정하는 xs-tools.iso 기본 제공 ISO 이미지에서 제공됩니다.

6.2.3. SUSE Enterprise Linux 10 SP1

이 설치에는 두 라운드 부팅 프로세스가 필요합니다. 첫 번째 라운드에서는 네트워크 저장소에서 커널 및 ramdisk를 다운로드하여 부팅합니다. 그런 다음 두 번째 라운드에서 디스크를 검사하여 설치된 커널 및 ramdisk를 찾고 PV-bootloader-args를 설정하여 게스트 파일 시스템 내부에 이 경로를 반영합니다. 이 프로세스는 SUSE가 pygrub 대신 사용하는 domUloader를 에뮬레이트합니다. 마지막으로 bootloader가 으로 설정되고 실행되어 정상적인 부팅을 시작합니다.

SLES 10 설치 메서드는 커널 및 ramdisk의 경로가 게스트 menu.lst 대신 VM 레코드에 저장됨을 의미하는데 YAST 패키지 관리자가 올바른 menu.lst를 쓰지 않기 때문에 이 방법이 작동을 가능하게 하는 유일한 방법입니다.

6.2.4. CentOS 4.5 / 5.0

CentOS 설치 메커니즘은 일부 MD5 체크섬이 eliloader가 인식하는 것과 다르다는 점을 제외하면 위에서 설명한 Red Hat 설치와 유사합니다.

6.3. VM에 Xenstore 항목 추가

개발자는 VM 유형에 따라 특별한 동작을 수행하는 게스트 에이전트를 VM에 설치하려고 할 수 있습니다. 이 정보를 게스트에 전달하기 위해 vm-data로 알려진 특별한 Xenstore 네임스페이스를 사용할 수 있습니다. 이 네임스페이스는 VM 생성 시 채워집니다. 이 네임스페이스는 VM 레코드의 맵에서 채워집니다.

VM에서 Xenstore 노드 foo를 채우려면:

1. VM 레코드에서 xenstore-data 매개 변수를 설정합니다.

   ```
   xe vm-param-set uuid=<vm_uuid> xenstore-data:vm-data/foo=bar
   ```

2. VM을 시작합니다.
3. Linux 기반 VM인 경우 XenServer Tools를 설치하고 xenstore-read를 사용하여 노드가 Xenstore에 있는지 확인합니다.

참고

vm-data로 시작하는 접두사만 허용되며 이 네임스페이스에 속하지 않은 항목은 VM을 시작할 때 모두 자동으로 무시됩니다.

6.4. 향상된 보안 기능

XenServer 7.1 이상의 제어 도메인에는 악의적인 게스트의 공격에 대비하기 위해 여러 가지 보안 기능이 향상되었습니다. 이러한 변경 내용으로 인해 올바른 기능이 손실되는 경우는 없지만 다른 배포에서 동작이 변경되므로 여기서 이러한 보안 향상 기능에 대해 설명합니다.

- libxenstore를 통해 액세스되는 소켓 인터페이스 xenstored. 인터페이스는 xs_restrict()로 제한됩니다.
- libxenctrl에서 xs_evtchn_open()을 호출하여 액세스하는 장치 /dev/xen/evtchn입니다. 핸들은 xs_evtchn_restrict()를 사용하여 제한할 수 있습니다.
- libxenctrl에서 xs_interface_open()을 통해 액세스하는 장치 /proc/xen/privcmd입니다. 핸들은 xc_interface_restrict()를 사용하여 제한됩니다. 임의의 하이퍼콜을 수행하는 기능 등 일부 권한 있는 명령은 원래 제한하기 어려우며 이러한 명령은 제한된 핸들에서만 금지됩니다.
- 제한된 핸들은 나중에 추가 권한을 부여받을 수 없으므로 인터페이스를 닫았다가 다시 열어야 합니다. 프로세스에서 향후에 추가로 핸들을 열 수 없는 경우에만 보안이 구현됩니다.

제어 도메인의 권한 있는 사용자 공간 인터페이스는 이제 특정 도메인에 대해서만 작동되도록 제한될 수 있습니다. 이러한 변경 내용의 영향을 받는 인터페이스에는 다음 세 가지가 있습니다.

- qemu 장치 에뮬레이션 프로세스 및 vncterm 터미널 에뮬레이션 프로세스는 루트가 아닌 사용자 ID로 실행되며 빈 디렉터리로 제한됩니다. 이 프로세스는 위 제한 API를 사용하여 가능한 경우 권한을 삭제합니다.
- xenstore에 대한 액세스는 악의적인 게스트로 인해 제어 도메인에서 서비스 거부가 발생하지 않도록 하기 위해 비율로 제한됩니다. 이 방법은 제한된 채우기 비율이 설정된 토큰 버킷으로 구현되는데 여기서 대부

분의 작업에는 토큰 한 개가 필요하며 트랜잭션을 여는 작업에는 토큰 20개가 필요합니다. 이 제한은 로드된 작업에서 동시에 많은 수의 게스트를 실행하는 경우에도 제한이 초과되지 않도록 높게 설정되었습니다.

- VNC 게스트 콘솔은 localhost 인터페이스에만 바인딩되므로 사용자가 개입하여 비활성화한 경우에도 제어 도메인 패킷 필터가 외부에 표시되지 않습니다.

6.5. 네트워크 인터페이스의 고급 설정

가상 및 물리적 네트워크 인터페이스에는 맵 매개 변수를 사용하여 구성할 수 있는 일부 고급 설정이 있습니다. 여기에는 일련의 사용자 지정 ethtool 설정과 일부 기타 설정이 있습니다.

6.5.1. ethtool 설정

개발자는 물리적 및 가상 네트워크 인터페이스에 대해 사용자 지정 ethtool 설정을 구성하려고 할 수 있습니다. 이러한 작업은 맵 매개 변수의 키를 사용하여 수행됩니다.

키	설명	유효한 설정
ethtool-rx	RX 체크섬 사용 여부 지정	설정을 사용하려는 경우 on 또는 true, 설정을 사용하지 않으려는 경우 off 또는 false
ethtool-tx	TX 체크섬 사용 여부 지정	설정을 사용하려는 경우 on 또는 true, 설정을 사용하지 않으려는 경우 off 또는 false
ethtool-sg	분산-수집 사용 여부 지정	설정을 사용하려는 경우 on 또는 true, 설정을 사용하지 않으려는 경우 off 또는 false
ethtool-tso	tcp 조각화 오프로드 사용 여부 지정	설정을 사용하려는 경우 on 또는 true, 설정을 사용하지 않으려는 경우 off 또는 false
ethtool-ufo	UDP 조각화 오프로드 사용 여부 지정	설정을 사용하려는 경우 on 또는 true, 설정을 사용하지 않으려는 경우 off 또는 false
ethtool-gso	일반 조각화 오프로드 사용 여부 지정	설정을 사용하려는 경우 on 또는 true, 설정을 사용하지 않으려는 경우 off 또는 false
ethtool-autoneg	자동 협상 사용 여부 지정	설정을 사용하려는 경우 on 또는 true, 설정을 사용하지 않으려는 경우 off 또는 false
ethtool-speed	Mb/초 단위로 장치 속도 설정	10, 100 또는 1000
ethtool-duplex	전이중 또는 반이중 모드 설정	half 또는 full

예를 들어 xe CLI를 사용하여 가상 NIC에서 TX 체크섬을 사용하려면 다음을 실행합니다.

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

또는

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

xe CLI를 사용하여 물리적 NIC의 이중 설정을 반이중으로 설정하려면 다음을 실행합니다.

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

6.5.2. 기타 설정

promiscuous 키를 on으로 설정하여 VIF 또는 PIF에서 무차별 모드를 설정할 수도 있습니다. 예를 들어 xe CLI를 사용하여 물리적 NIC에서 무차별 모드를 사용하려면 다음을 실행합니다.

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

또는

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

VIF 및 PIF 개체에는 읽기 전용이며 인터페이스에 대한 최대 전송 단위의 현재 설정을 제공하는 MTU 매개 변수가 있습니다. other-config 맵 매개 변수의 mtu 키를 사용하여 물리적 또는 가상 NIC의 기본 최대 전송 단위를 재정의할 수 있습니다. 예를 들어 가상 NIC에서 MTU를 다시 설정하려면 xe CLI를 통해 점보 프레임을 사용합니다.

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

기본 인터페이스의 MTU 변경은 실험적인 고급 기능으로, 단일 리소스 풀의 여러 NIC에 다양한 MTU가 있는 경우 예기치 않은 파생 작업이 발생할 수 있습니다.

6.6. SR 이름 국제화

설치 시 생성된 SR에는 이제 해당 이름을 국제화하는 방법을 나타내는 other_config 키가 있습니다.

other_config["i18n-key"]는 다음 중 하나가 될 수 있습니다.

- local-hotplug-cd
- local-hotplug-disk
- local-storage
- xenserver-tools

또한 other_config["i18n-original-value-<field name>"]는 SR이 생성되었을 당시의 필드 값을 제공합니다. XenCenter에서 이 과 같은 레코드를 확인하는 경우(즉, XenServer 설치 과정에서 생성된 이래 레코드가 변경되지 않음) 국제화가 적용됩니다. 즉, XenCenter는 해당 필드의 현재 내용을 무시하고 대신 사용자 고유 언어에 적절한 값을 사용합니다.

사용자 특정 목적을 위해 SR.name_label을 변경하는 경우에는 더 이상 other_config["i18n-original-value-name_label"]과 같지 않습니다. 따라서 XenCenter는 국제화를 적용하지 않고 사용자가 지정한 이름을 유지합니다.

6.7. XenCenter에서 개체 숨기기

HideFromXenCenter=true 키를 개체의 other_config 매개 변수에 추가하여 XenCenter에서 네트워크, PIF 및 VM을 숨길 수 있습니다. 이 기능은 수행하는 작업에 대한 확실한 지식이 있는 ISV를 위한 것으로, 일반 사용자를 위한 것이 아닙니다. 예를 들어 사용자 환경의 일반 사용자가 직접 사용해서는 안 되는 복제된 VM을 숨기려고 할 수 있습니다.

XenCenter에서 View(보기) 메뉴를 사용하면 숨겨진 네트워크, PIF 및 VM을 표시할 수 있습니다.

CITRIX

7장. XenCenter API 확장

다음 섹션에서는 문서화된 API 외에 가정 및 API 확장에 대해 자세히 설명합니다. 확장은 VM.other_config와 같이 사전에 특정 키-값 쌍으로 인코딩됩니다.

7.1. 풀

키	의미
pool.name_label	빈 name_label은 풀을 트리 보기에서 숨겨야 함을 나타냅니다.
pool.rolling_upgrade_in_progress	풀이 롤링 업그레이드 중에 있는 경우 나타납니다.

7.2. 호스트

키	의미
host.other_config["iscsi_iqn"]	호스트의 iSCSI IQN입니다.
host.license_params["expiry"]	호스트 라이센스의 만료 날짜입니다(ISO 8601, UTC).
host.license_params["sku_type"]	호스트 라이센스 유형입니다(예: Server 또는 Enterprise).
host.license_params["restrict_pooling"]	호스트에서 풀링을 제한한 경우 true를 반환합니다.
host.license_params["restrict_connection"]	XenCenter에서 만들 수 있는 연결 수가 제한됩니다.
host.license_params["restrict_qos"]	호스트에서 서비스 품질 설정이 활성화된 경우 true를 반환합니다.
host.license_params["restrict_vlan"]	호스트에서 가상 네트워크 생성이 제한된 경우 true를 반환합니다.
host.license_params["restrict_pool_attached_storage"]	이 호스트에서 공유 스토리지 생성이 제한된 경우 true를 반환합니다.
host.software_version["product_version"]	호스트의 제품 버전을 반환합니다.
host.software_version["build_number"]	호스트의 빌드 번호를 반환합니다.
host.software_version["xapi"]	호스트의 API 수정 번호를 반환합니다.
host.software_version["package-linux"]	Linux 팩이 설치되어 있는 경우 "installed"를 반환합니다.
host.software_version["oem_build_number"]	호스트가 OEM 버전이면 수정 번호를 반환합니다.

키	의미
host.logging["syslog_destination"]	XenServer 시스템 로거의 대상을 가져오거나 설정합니다(로컬 로깅의 경우 null).
host.logging["multipathing"]	이 호스트에서 스토리지 다중 경로가 활성화된 경우 "true"입니다.
host.logging["boot_time"]	호스트가 부팅된 시간을 지정하는 부동 소수점 Unix 시간입니다.
host.logging["agent_start_time"]	제어 도메인 관리 디먼이 시작된 시간을 지정하는 부동 소수점 Unix 시간입니다.

7.3. VM

키	의미
VM.other_config["default_template"]	이 템플릿은 Citrix에서 설치된 템플릿입니다. 이 템플릿은 트리 보기에서 선택적으로 숨기고, 다른 아이콘을 사용하고, 삭제를 금지하는 데 사용됩니다.
VM.other_config["xensource_internal"]	이 템플릿은 P2V 서버 템플릿과 같이 특수한 템플릿으로 UI에서 완전히 숨겨집니다.
VM.other_config["install_distro"] == "rhlike"	이 템플릿은 RHEL 4.5, RHEL 5 또는 이와 동일한 CentOS 버전용입니다. Miami의 ISO/CD에서의 설치에 대한 지원을 포함하여 설치 중 설치 저장소에 대한 프롬프트를 표시하고 이 설치 관리자에 맞도록 NFS URL을 수정하는 데 사용됩니다.
VM.other_config["install_distro"] in { "rhel41" | "rhel44" }	이 템플릿은 RHEL 4.1, RHEL 4.4 또는 이와 동일한 CentOS 버전용입니다. 설치 중 설치 저장소에 대한 프롬프트를 표시하고 이 설치 관리자에 맞도록 NFS URL을 수정하는 데 사용됩니다. 이 템플릿에 대한 ISO 지원은 사용할 수 없습니다.
VM.other_config["install_distro"] == "sleslike"	이 템플릿은 SLES 10 및 SLES 9용입니다. EL5의 경우처럼 설치 중 설치 저장소에 대한 프롬프트를 표시하는 데 사용되지만 이 경우 NFS URL은 수정되지 않습니다. XenServer 7.1에서는 SLES 10에 대한 ISO 지원이 제공되지 않습니다. 해당 플랫폼에서 install-methods를 사용하여 SLES 9와 SLES 10을 구분하십시오.
VM.other_config["install-repository"] == "cdrom"	URL 대신 VM의 연결된 CD 드라이브의 저장소에서 설치하도록 요청합니다.

키	의미
VM.other_config["auto_poweron"]	서버 부팅 시 VM이 시작되는지 여부를 가져오거나 설정합니다. "true" 또는 "false"입니다.
VM.other_config["ignore_excessive_vcpus"]	해당 호스트가 가지고 있는 물리적 CPU 수보다 VM의 VCPU 수가 많은 경우 XenCenter's 경고를 무시할지 여부를 가져오거나 설정합니다. 무시하는 경우 true입니다.
VM.other_config["HideFromXenCenter"]	XenCenter에서 트리 보기에 VM을 표시할지 여부를 가져오거나 설정합니다. 숨기는 경우 "true"입니다.
VM.other_config["import_task"]	이 VM을 만든 가져오기 작업을 가져옵니다.
VM.HVM_boot_params["order"]	HVM VM에 대해서만 VM의 부팅 순서를 가져오거나 설정합니다. 예를 들어 "CDN"은 제일 먼저 디스크를 부팅하고, 그 다음 CD 드라이브, 네트워크 순으로 부팅합니다.
VM.VCPU_params["weight"]	VM의 VCPU에 대한 IONice 값을 가져오거나 설정합니다. 값은 1에서 65536 사이이며 최대값은 65536입니다.
VM.pool_migrate(..., options['live'])	true는 라이브 마이그레이션을 나타냅니다. XenCenter에서는 항상 이 값을 사용합니다.
VM.other_config["install-methods"]	이 템플릿에 사용할 수 있는 설치 방법에 대한 쉼표로 구분된 목록입니다. "cdrom", "nfs", "http" 또는 "ftp"를 포함할 수 있습니다.
VM.other_config["last_shutdown_time"]	이 VM이 마지막으로 종료되었거나 다시 부팅된 시간으로 UTC ISO8601 날짜 및 시간 형식을 사용합니다.
VM.other_config["p2v_source_machine"]	P2V 프로세스에서 이 VM을 가져온 경우 원본 컴퓨터입니다.
VM.other_config["p2v_import_date"]	P2V 프로세스에서 가져온 경우 VM을 가져온 날짜입니다. UTC ISO8601 날짜 및 시간 형식을 사용합니다.

7.4. SR

키	의미
SR.other_config["auto-scan"]	자동으로 SR에서 변경 내용을 검사합니다. XenCenter에서 생성된 모든 SR에 설정합니다.
SR.sm_config["type"]	물리적 CD 드라이브인 SR에 대해서는 cd 유형으로 설정합니다.

7.5. VDI

키	의미
VDI.type	system 대신 user가 "이 디스크가 VM에 연결되어 있는 경우 GUI를 통한 VDI 삭제를 허용하거나 허용하지 않음"을 의미하는 데 사용됩니다. 여기서의 의도는 VM 손상을 방지하는 것입니다(대신 제거해야 함). suspend 및 crashdump는 각각 일시 중단 및 코어 덤프를 기록합니다. ephemeral은 현재 사용되지 않습니다.
VDI.managed	관리되지 않는 모든 VDI가 UI에서 완전히 숨겨집니다. VHD 체인의 분기점이거나 사용되지 않은 VDI당 LUN 디스크입니다.
VDI.sm_config["vmhint"]	이 VDI가 지원하는 VM의 UUID입니다. 이는 사용자 인터페이스를 통해 VDI를 생성할 때 특정 스토리지 백엔드의 성능을 향상하려는 경우 설정합니다.

7.6. VBD

키	의미
VBD.other_config["is_owner"]	설정하면 VM이 제거될 때 이 디스크가 삭제될 수 있습니다.
VBD.other_config["class"]	ionice의 Best Effort 설정에 해당하는 정수로 설정합니다.

7.7. 네트워킹

키	의미
network.other_config["automatic"]	이 키에 false 이외의 값이 있는 경우 New VM(새 VM) 마법사는 기본적으로 이 네트워크에 연결된 VIF를 만듭니다.
network.other_config["import_task"]	이 네트워크를 만든 가져오기 작업을 가져옵니다.

7.8. VM_guest_metrics

키	의미
PV_drivers_version["major"]	VM의 PV 드라이버 버전의 주 버전을 가져옵니다.
PV_drivers_version["minor"]	VM의 PV 드라이버 버전의 부 버전을 가져옵니다.
PV_drivers_version["micro"]	VM의 PV 드라이버 버전의 마이크로 버전(빌드 번호)을 가져옵니다.

7.9. 작업

키	의미
task.other_config["object_creation"] == "complete"	VM 가져오기와 관련된 작업의 경우 이 플래그는 모든 개체(VM, 네트워크)가 생성된 경우 설정됩니다. Import VM(VM 가져오기) 마법사에서 이를 필요로 하는 모든 네트워크를 찾아 다시 매핑하는 데 이 플래그가 유용할 수 있습니다.