Red Hat Virtualization 4.4

Ruby SDK 가이드

Red Hat Virtualization Ruby SDK 사용

Last Updated: 2022-06-26

Red Hat Virtualization 4.4 Ruby SDK 가이드

Red Hat Virtualization Ruby SDK 사용

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

법적 공지

초록

이 가이드에서는 Red Hat Virtualization Ruby 소프트웨어 개발 키트를 설치하고 사용하는 방법을 설명합니다.

차례

1장. 개요

참고

Ruby 소프트웨어 개발 키트(SDK)는 더 이상 사용되지 않습니다. Ruby SDK에 대한 지원은 이후 릴리스에서 제거될 예정입니다.

Ruby 소프트웨어 개발 키트는 Ruby gem으로 Ruby 프로젝트의 Red Hat Virtualization Manager와 상호 작용할 수 있습니다. 이러한 클래스를 다운로드하여 프로젝트에 추가하면 관리 작업의 고급 자동화를 위해 다양한 기능에 액세스할 수 있습니다.

1.1. 사전 요구 사항

Ruby 소프트웨어 개발 키트를 설치하려면 다음이 있어야 합니다.

- Red Hat Enterprise Linux 8이 설치된 시스템입니다. Server 및 Workstation 변형이 모두 지원됩니다.
- Red Hat Virtualization 인타이틀먼트 서브스크립션.

1.2. RUBY SOFTWARE DEVELOPMENT KIT 설치

1. 필요한 리포지토리를 활성화합니다.

```
# subscription-manager repos \
    --enable=rhel-8-for-x86_64-baseos-rpms \
    --enable=rhel-8-for-x86_64-appstream-rpms \
    --enable=rhv-4.4-manager-for-rhel-8-x86_64-rpms
```

2. Ruby Software Development Kit를 설치합니다.

```
# dnf install rubygem-ovirt-engine-sdk4
```

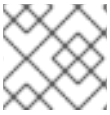
또는 **gem** 으로 설치할 수 있습니다 :

```
# gem install ovirt-engine-sdk
```

1.3. 종속 항목

Ruby 소프트웨어 개발 키트에는 다음과 같은 종속성이 있으며 **gem** 을 사용하는 경우 수동으로 설치해야 합니다.

- XML 구문 분석 및 렌더링을 위한 **libxml2**
- HTTP 전송용 **libcurl**
- C 컴파일러
- 필수 헤더 및 라이브러리 파일

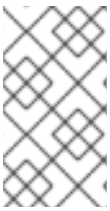


참고

RPM을 설치한 경우 종속성 파일을 설치할 필요가 없습니다.

종속성 파일을 설치합니다.

```
# dnf install gcc libcurl-devel libxml2-devel ruby-devel
```



참고

Debian 또는 Ubuntu를 사용하는 경우 **apt-get** 을 사용하십시오.

```
# apt-get install gcc libxml2-dev libcurl-dev ruby-dev
```

2장. 소프트웨어 개발 키트 사용

이 장에서는 Ruby 소프트웨어 개발 키트의 모듈과 클래스를 정의하고 사용법을 설명합니다.

2.1. 클래스

OvirtSDK4 모듈에는 다음 소프트웨어 개발 키트 클래스가 포함되어 있습니다.

연결

연결 클래스는 서버에 연결하고 서비스 트리의 루트에 대한 참조를 얻는 메커니즘입니다. 자세한 내용은 서버에 연결을 참조하십시오.

유형

Type 클래스는 API에서 지원하는 유형을 구현합니다. 예를 들어, Vm 클래스는 가상 머신 유형의 구현입니다. 클래스는 데이터 컨테이너이며 논리를 포함하지 않습니다. 유형 인스턴스를 사용하게 됩니다. 이러한 클래스의 인스턴스는 매개 변수로 사용되며 서비스 메서드의 값을 반환합니다. 기본 표현으로 전환하거나 기본 표현으로의 변환은 소프트웨어 개발 키트에 의해 투명하게 처리됩니다.

서비스

Service 클래스는 API에서 지원하는 서비스를 구현합니다. 예를 들어, VmsService 클래스는 시스템의 가상 머신 컬렉션을 관리하는 서비스 구현입니다.
이러한 클래스의 인스턴스는 서비스를 참조할 때 SDK에 의해 자동으로 생성됩니다. 예를 들어 **SystemService** 클래스의 **vms_service** 메서드를 호출할 때 SDK에 의해 **VmsService** 클래스의 새 인스턴스가 자동으로 생성됩니다.

```
vms_service = connection.system_service.vms_service
```

주의

이러한 클래스의 인스턴스를 수동으로 생성하지 마십시오. 생성자 매개변수 및 메서드는 나중에 변경될 수 있습니다.

오류

Error 클래스는 오류를 보고할 때 소프트웨어 개발 키트가 발생하는 기본 예외 클래스입니다.
특정 특정 오류 클래스는 기본 오류 클래스를 확장합니다.

- AuthError - 인증 또는 권한 부여 오류
- ConnectionError - 서버 이름을 확인할 수 없거나 서버에 연결할 수 없음
- NotFoundError - 요청 된 개체가 없습니다.
- TimeoutError - 작업 시간 제한

기타 클래스

기타 클래스(예: HTTP 클라이언트 클래스, 리더 및 작성자)는 HTTP 통신 및 XML 구문 분석 및 렌더링에 사용됩니다. 나중에 변경될 수 있는 내부 구현 세부 정보가 포함되어 있으므로 이러한 클래스를 사용하는 것은 권장되지 않습니다. 이전 버전과의 호환성은 신뢰할 수 없습니다.

2.2. 유형

2.2.1. 유형 인스턴스 생성 및 수정

아래에 설명된 대로 변경 사항이 서비스 메서드를 호출하는 서버로 명시적으로 전송되지 않는 한 형식 인스턴스를 만들거나 수정해도 서버 쪽에 영향을 미치지 않습니다.Creating or modifying an instance of a type does not have any effect on the server side, unless the changes are explicitly sent to the server calling a service method, as described below. 서버 측의 변경 사항은 이미 메모리에 이미 존재하는 인스턴스에 자동으로 반영되지 않습니다.

이러한 클래스의 생성자에는 형식의 각 속성에 대해 여러 선택적 인수가 있습니다.The constructors of these classes have multiple optional arguments, one for each attribute of the type. 이는 여러 생성자에 대한 중첩된 호출을 사용하여 오브젝트 생성을 간소화합니다.

다음 예에서는 가상 머신 인스턴스가 생성되고 해당 속성(클러스터, 템플릿 및 메모리)이 설정됩니다.

특성을 사용하여 가상 머신 인스턴스 생성

```
vm = OvirtSDK4::Vm.new(
  name: 'myvm',
  cluster: OvirtSDK4::Cluster.new(
    name: 'mycluster'
  ),
  template: OvirtSDK4::Template.new(
    name: 'mytemplate'
  ),
  memory: 1073741824
)
```

해시(예: **cluster) 이러한 생성자에 전달된 OvirtSDK4::Cluster.new**는 재귀적으로 처리됩니다.

다음 예제에서 클러스터 및 템플릿 클래스의 생성자를 명시적으로 호출하는 대신 일반 해시가 사용됩니다. SDK는 내부적으로 해시를 필요한 클래스로 변환합니다.

Attributes Expressed as Plain Hashes를 사용하여 가상 머신 인스턴스 생성

```
vm = OvirtSDK4::Vm.new(
  name: 'myvm',
  cluster: {
    name: 'mycluster'
  },
  template: {
    name: 'mytemplate'
  },
  memory: 1073741824
)
```

이러한 방식으로 생성자를 사용하는 것이 권장되지만 필수는 아닙니다.

다음 예제에서는 생성자를 호출할 때 인수 없이 가상 머신 인스턴스가 생성됩니다. setters를 사용하거나 setter 및 constructors 조합을 사용하여 가상 머신 인스턴스의 속성을 하나씩 추가할 수 있습니다.

가상 머신 인스턴스 생성 및 개별 속성 추가

```
vm = OvirtSDK4::Vm.new
vm.name = 'myvm'
vm.cluster = OvirtSDK4::Cluster.new(name: 'mycluster')
vm.template = OvirtSDK4::Template.new(name: 'mytemplate')
vm.memory = 1073741824
```

API 사양의 오브젝트 목록으로 정의된 속성은 Ruby 배열로 구현됩니다. 예를 들어 **Vm** 유형의 **custom_properties** 속성은 **CustomProperty** 유형의 개체 목록으로 정의됩니다.

특성 목록을 배열로 추가Adding a List of attributes as an Array

```
vm = OvirtSDK4::Vm.new(
  name: 'myvm',
  custom_properties: [
    OvirtSDK4::CustomProperty.new(...),
    OvirtSDK4::CustomProperty.new(...),
    ...
  ]
)
```

API에서 열거된 값으로 정의된 속성은 열거된 형식과 동일한 이름의 모듈에서 상수로 구현됩니다.

다음 예제에서는 **VmStatus** enumerated 값을 사용하여 **Vm** 유형의 status 속성을 정의하는 방법을 보여줍니다.

```
case vm.status
when OvirtSDK4::VmStatus::DOWN
  ...
when OvirtSDK4::VmStatus::IMAGE_LOCKED
  ...
end
```

중요

API 사양에서 enum 형식 값은 XML 및 JSON에 대한 규칙이므로 소문자입니다.In the API specification, the values of **enum** types are lower case, because that is the convention for XML and JSON. 그러나 Ruby에서 규칙은 이러한 상수에 대해 **대문자** 를 사용하는 것입니다.

2.2.2. 인스턴스 속성 검색

해당 특성 판독기를 사용하여 인스턴스 속성을 검색할 수 있습니다.

다음 예제에서는 가상 머신 인스턴스의 이름 및 메모리를 검색합니다.

가상 머신 인스턴스 속성 검색

```
puts "vm.name: #{vm.name}"
puts "vm.memory: #{vm.memory}"
```

```
vm.custom_properties.each do |custom_property|
  ...
end
```

인스턴스 속성을 링크로 검색

일부 인스턴스 속성은 링크로 반환되며, 데이터를 검색하려면 **follow_link** 메서드가 필요합니다. 다음 예에서 가상 머신의 속성에 대한 요청에 대한 응답은 링크를 사용하여 XML로 포맷됩니다.

가상 머신 속성을 링크로 검색

```
<vm id="123" href="/ovirt-engine/api/vms/123">
  <name>myvm</name>
  <link rel="diskattachments" href="/ovirt-engine/api/vms/123/diskattachments/">
  ...
</vm>
```

링크 **vm.disk_attachments** 에는 실제 디스크 첨부 파일이 포함되어 있지 않습니다. 데이터를 검색하기 위해 Connection 클래스는 **href** XML 속성의 값을 사용하여 실제 데이터를 검색하는 follow_link 메서드를 제공합니다.

다음 예제에서 **follow_link** 를 사용하면 디스크 첨부 파일로 이동한 다음 각 디스크로 이동하여 **별칭** 을 검색할 수 있습니다.

가상 머신 서비스 검색

```
vm = vm_service.get
```

follow_link 를 사용하여 디스크 연결 검색 및 디스크 별칭

```
attachments = connection.follow_link(vm.disk_attachments)
attachments.each do |attachment|
  disk = connection.follow_link(attachment.disk)
  puts "disk.alias: #{disk.alias}"
end
```

2.3. 서비스

2.3.1. 서비스 검색

API는 서버 경로와 연결된 서비스 세트를 제공합니다. 예를 들어 시스템의 가상 머신 컬렉션을 관리하는 서비스는 **/vms** 에 있으며 식별자가 **123** 인 가상 시스템을 관리하는 서비스는 **/vms/123** 에 있습니다.

Ruby 소프트웨어 개발 키트에서 해당 서비스 트리의 루트는 **시스템 서비스에** 의해 구현됩니다. 연결의 system_service 메서드를 호출하여 이를 검색합니다.

시스템 서비스 검색

```
system_service = connection.system_service
```

시스템 서비스에 대한 참조가 있으면 ***_service** 메서드(Service **locators**라고도 함)를 사용하여 다른 서비스에 대한 참조를 검색하는 데 사용할 수 있습니다.

예를 들어 시스템의 가상 머신 컬렉션을 관리하는 서비스에 대한 참조를 검색하려면 vms_service 서비스 locator를 사용할 수 있습니다.

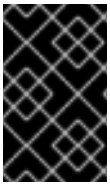
기타 서비스 검색

```
vms_service = system_service.vms_service
```

식별자가 **123** 인 가상 머신을 관리하는 서비스에 대한 참조를 검색하려면 vm_service 서비스의 서비스 로케이터를 사용합니다. 서비스 로케이터는 가상 머신 식별자를 매개 변수로 사용합니다.

식별자를 사용하여 가상 머신 서비스 검색

```
vm_service = vms_service.vms_service('123')
```



중요

서비스 로케이터 호출에서 반환된 오브젝트는 순수 서비스이며 데이터를 포함하지 않습니다. 예를 들어 이전 예제에서 검색된 **vm_service** Ruby 오브젝트는 가상 시스템의 표현이 아닙니다. 가상 머신을 검색, 업데이트, 삭제, 시작 및 중지하는 데 사용되는 서비스입니다.

2.3.2. 서비스 방법

원하는 서비스를 찾은 후 서비스 메서드를 호출할 수 있습니다. 이러한 메서드는 서버에 요청을 보내고 실제 작업을 수행합니다.

오브젝트 컬렉션을 관리하는 서비스에는 일반적으로 메서드 **목록** 및 **추가** 방법이 있습니다.

단일 오브젝트를 관리하는 서비스에는 일반적으로 **get**,**update** 및 **remove** 메서드가 있습니다.

서비스에는 검색, 생성, 업데이트 또는 제거 이외의 작업을 수행하는 추가 작업 방법이 있을 수 있습니다. 이러한 메서드는 단일 개체를 관리하는 서비스에서 가장 일반적으로 찾을 수 있습니다.

2.3.2.1. Get

get 메서드는 단일 개체의 표현을 검색합니다.

다음 예제에서는 식별자가 **123** 인 가상 머신의 표현을 찾아서 검색합니다.

```
# Find the service that manages the virtual machine:
vms_service = system_service.vms_service
vm_service = vms_service.vm_service('123')

# Retrieve the representation of the virtual machine:
vm = vm_service.get
```

결과는 해당 유형의 인스턴스입니다. 이 경우 결과는 Ruby 클래스 Vm 의 인스턴스입니다.

일부 서비스의 **get** 메서드는 개체의 표현을 검색하는 방법 또는 하나 이상의 경우 검색할 표현을 제어하는 추가 매개 변수를 지원합니다.

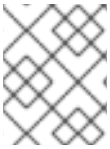
예를 들어 부팅 후 가상 머신의 향후 상태를 검색할 수 있습니다. 가상 머신을 관리하는 서비스의 **get** 메서드는 next_run 부울 매개 변수를 지원합니다.

가상 머신의 next_run 상태 검색

```
# Retrieve the representation of the virtual machine; not the
# current one, but the one that will be used after the next
# boot:
vm = vm_service.get(next_run: true)
```

자세한 내용은 소프트웨어 개발 키트 참조 문서를 참조하십시오.

개체를 검색할 수 없는 경우 소프트웨어 개발 키트는 오류에 대한 세부 정보가 포함된 Error 예외를 발생시킵니다. 존재하지 않는 오브젝트를 검색하려고 하면 이 문제가 발생합니다.



참고

서비스 locator 메서드가 서버에 요청을 보내지 않기 때문에 오브젝트가 없는 경우에도 **서비스 로케이터** 메서드 호출이 실패하지 않습니다.

다음 예제에서는 get 메서드의 예외를 발생시키는 동안 서비스 로케이터 메서드가 성공합니다.In the following examples, the **service locator** method will succeed, while the **get** method will raise an exception:

존재하지 않는 가상 머신의 서비스 찾기: 오류 없음

```
# Find the service that manages a virtual machine that does
# not exist. This will succeed.
vm_service = vms_service.vm_service('non_existent_VM')
```

존재하지 않는 가상 머신 서비스 검색: 오류

```
# Retrieve the virtual machine. This will raise an exception.
vm = vm_service.get
```

2.3.2.2. list

list 메서드는 컬렉션에서 여러 개체의 표현을 검색합니다.

가상 머신 컬렉션 나열

```
# Find the service that manages the collection of virtual
# machines:
vms_service = system_service.vms_service
vms = vms_service.list
```

결과는 해당 유형의 인스턴스를 포함하는 Ruby 배열입니다. 위의 예에서 응답은 Ruby 클래스 Vm 의 인스턴스 목록입니다.

일부 서비스의 **목록** 방법은 추가 매개 변수를 지원합니다.

예를 들어 거의 모든 최상위 컬렉션은 **검색** 매개 변수를 지원하여 결과를 필터링하고 **max** 매개 변수는 서버에서 반환된 결과 수를 제한합니다.

Ten Virtual Machines calleded "my*" 나열

```
vms = vms_service.list(search: 'name=my*', max: 10)
```

참고

모든 목록 방법이 **search** 또는 **max** 매개변수를 지원하는 것은 아닙니다. 일부 목록 방법은 다른 매개변수를 지원할 수 있습니다. 자세한 내용은 참조 문서를 참조하십시오.

결과 목록이 비어 있으면 반환된 값은 빈 Ruby 배열입니다. 절대로 **nil** 이 될 수 없습니다.

결과 목록을 검색할 수 없는 경우 SDK는 실패 세부 정보가 포함된 오류 예외를 발생시킵니다.

2.3.2.3. add

메서드를 추가하면 컬렉션에 새 요소를 추가합니다. **추가할 오브젝트를 설명하는 관련 유형의 인스턴스를 수신하고, 추가할 요청을 보내고, 추가된 오브젝트를 설명하는 유형의 인스턴스를 반환합니다.**

새 가상 머신 추가

```
# Add the virtual machine:
vm = vms_service.add(
  OvirtSDK4::Vm.new(
    name: 'myvm',
    cluster: {
      name: 'mycluster'
    },
    template: {
      name: 'mytemplate'
    }
  )
)
```

중요

add 메서드에서 반환된 Ruby 오브젝트는 관련 유형의 인스턴스입니다. 이는 서비스가 아니며 데이터의 컨테이너일 뿐입니다. 위의 예에서 반환된 오브젝트는 Vm 클래스의 인스턴스입니다.

방금 추가한 가상 머신에서 작업을 수행해야 하는 경우 이를 관리하고 서비스 로케이터를 호출하는 서비스를 찾아야 합니다.

새 가상 머신 시작

```
# Add the virtual machine:
vm = vms_service.add(
  ...
)

# Find the service that manages the virtual machine:
vm_service = vms_service.vm_service(vm.id)

# Start the virtual machine:
vm_service.start
```

대부분의 개체 생성은 비동기 작업입니다. 예를 들어 새 가상 머신을 생성하는 경우 가상 머신을 완전히 생성하고 사용할 준비가 되기 전에 **add** 메서드는 가상 머신을 반환합니다. 완전히 생성될 때까지 오브젝트의 상태를 폴링해야 합니다. 가상 머신의 경우 상태가 **DOWN** 이 될 때까지 확인합니다.

가상 시스템을 생성하고, 새 가상 시스템을 관리하는 서비스를 찾고, 가상 머신 상태가 **DOWN** 이 될 때까지 상태를 반복적으로 검색한 후 모든 디스크가 생성되었음을 나타내는 것이 좋습니다.

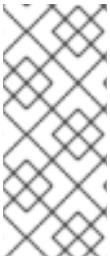
가상 머신 추가, 서비스 가져오기 및 상태 검색

```
# Add the virtual machine:
vm = vms_service.add(
  ...
)

# Find the service that manages the virtual machine:
vm_service = vms_service.vm_service(vm.id)

# Wait until the virtual machine is DOWN, indicating that all the
# disks have been created:
loop do
  sleep(5)
  vm = vm_service.get
  break if vm.status == OvirtSDK4::VmStatus::DOWN
end
```

개체를 생성할 수 없는 경우 **SDK**는 오류 세부 정보가 포함된 **Error** 예외를 발생시킵니다. **nil** 을 반환하지 않습니다.

2.3.2.4. update

메서드를 업데이트하여 기존 오브젝트를 업데이트합니다. 수행할 업데이트를 설명하고, 업데이트를 위해 요청을 보내고, 업데이트된 오브젝트를 설명하는 유형의 인스턴스를 반환합니다.



참고

이 업데이트 방법에서 반환된 **Ruby** 오브젝트는 관련 유형의 인스턴스입니다. 이는 서비스가 아니며 데이터의 컨테이너일 뿐입니다. 이 특정 예에서 반환된 오브젝트는 **Vm** 클래스의 인스턴스입니다.

다음 예에서 서비스 로케이터 메서드는 가상 머신을 관리하는 서비스를 찾고 업데이트 방법은 해당 이름을 업데이트합니다.

가상 머신 이름 업데이트

```
# Find the virtual machine and the service that
# manages it:
vm = vms_service.list(search: 'name=myvm').first
vm_service = vms_service.vm_service(vm.id)

# Update the name:
updated_vm = vms_service.update(
  OvirtSDK4::Vm.new(
    name: 'newvm'
  )
)
```

오브젝트를 업데이트할 때 업데이트하려는 속성만 업데이트합니다.

가상 머신의 선택한 속성 업데이트(권장)

```
vm = vm_service.get
vm.name = 'newvm'
```

전체 오브젝트를 업데이트하지 마십시오.

가상 머신의 모든 속성 업데이트(권장 없음)

```
# Retrieve the current representation:
vms_service.update(vm)
```

가상 머신의 모든 속성을 업데이트하면 리소스가 낭비되며 서버 측에서 예기치 않은 버그가 발생할 수 있습니다.

일부 서비스의 업데이트 방법은 업데이트 방법이나 업데이트를 제어하는 데 사용할 수 있는 추가 매개 변수를 지원합니다. 예를 들어 현재 상태가 아닌 가상 머신의 메모리를 업데이트할 수 있지만 다음에 시작할 수 있습니다. 가상 머신을 관리하는 서비스의 업데이트 방법은 next_run 부울 매개 변수를 지원합니다.

다음 실행 시 가상 머신의 메모리 업데이트

```
vm = vm_service.update(
  OvirtSDK4::Vm.new(
    memory: 1073741824
  ),
  next_run: true
)
```

업데이트를 수행할 수 없는 경우 SDK는 오류 세부 정보가 포함된 Error 예외를 발생시킵니다. nil 을 반환하지 않습니다.

2.3.2.5. remove

기존 오브젝트를 제거하는 메서드를 제거합니다. 일반적으로 매개 변수는 단일 오브젝트를 관리하는

서비스 방법이며 서비스는 제거할 오브젝트를 이미 알고 있기 때문에 매개 변수를 지원하지 않습니다.

식별자가 123인 가상 머신 제거

```
vm_service = vms_service.vm_service('123')
vms_service.remove
```

메서드 제거 방법 또는 제거 방법을 제어하는 매개 변수를 지원합니다. 예를 들어 detach_only 부울 매개 변수를 사용하여 디스크를 유지하면서 가상 머신을 제거할 수 있습니다.

디스크를 예약하는 동안 가상 머신 제거

```
vm_service.remove(detach_only: true)
```

remove 메서드는 개체가 성공적으로 제거되면 nil 을 반환합니다. 삭제된 개체를 반환하지 않습니다.

오브젝트를 제거할 수 없는 경우 SDK는 오류 세부 정보가 포함된 Error 예외를 발생시킵니다.

2.3.2.6. 추가 작업

위에서 설명한 방법 외에도 추가 조치 방법이 있습니다. 가상 시스템을 관리하는 서비스에는 이를 시작하고 중지할 수 있습니다.

가상 머신 시작

```
vm_service.start
```

일부 작업 방법에는 작업을 수정하는 매개변수가 있습니다. 예를 들어 시작 방법은 use_cloud_init 매개변수를 지원합니다.

Cloud-Init를 사용하여 가상 머신 시작

```
vm_service.start(use_cloud_init: true)
```

대부분의 작업 메서드가 성공하면 nil 을 반환하고 오류가 발생하면 오류가 발생합니다. 그러나 일부 작업 메서드는 값을 반환합니다. 예를 들어 스토리지 도메인을 관리하는 서비스에는 스토리지 도메인이 이미 데이터 센터에 연결되어 있는지 확인하는 is_attached 작업 메서드가 있습니다. is_Attached 작업 메서드는 부울 값을 반환합니다.

연결된 스토리지 도메인 확인

```
sds_service = system_service.storage_domains_service
sd_service = sds_service.storage_domain_service('123')
if sd_service.is_attached
  ...
end
```

각 서비스, 매개변수 및 반환 값에서 지원하는 작업 방법을 확인하려면 소프트웨어 개발 키트의 참조 문서를 참조하십시오.

3장. RUBY 예

3.1. RED HAT VIRTUALIZATION MANAGER에 연결

연결 클래스는 소프트웨어 개발 키트의 진입점입니다. Red Hat Virtualization Manager REST API의 서비스에 대한 액세스를 제공합니다.

Connection 클래스의 매개 변수는 다음과 같습니다.

- URL - Red Hat Virtualization Manager API의 기본 URL
- 사용자 이름
- 암호
- ca_file - 신뢰할 수 있는 CA 인증서가 포함된 PEM 파일. TLS로 보호되는 서버에 연결할 때 ca.pem 파일이 필요합니다. ca_file 을 지정하지 않으면 시스템 전체 CA 인증서 저장소가 사용됩니다.

Red Hat Virtualization Manager에 연결

```
connection = OvirtSDK4::Connection.new(
  url: 'https://engine.example.com/ovirt-engine/api',
  username: 'admin@internal',
  password: '...',
  ca_file: 'ca.pem',
)
```

중요

연결에는 서버에 대한 **HTTP** 연결 풀 및 인증 토큰을 포함하여 중요한 리소스가 있습니다. 이러한 리소스를 더 이상 사용하지 않는 경우 해제해야 합니다.

```
connection.close
```

연결 및 해당 연결에서 얻은 모든 서비스는 연결이 종료된 후에는 사용할 수 없습니다.

연결에 실패하면 소프트웨어 개발 키트가 오류에 대한 세부 정보가 포함된 오류 예외를 발생시킵니다.

자세한 내용은 **Connection:initialize** 을 참조하십시오.

3.2. 데이터 센터 나열

이 **Ruby** 예제에서는 데이터 센터를 나열합니다.

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Get the reference to the service that manages the
# collection of data centers:
dcs_service = system_service.data_centers_service

# Retrieve the list of data centers and for each one
# print its name:
dcs = dcs_service.list
dcs.each do |dc|
  puts dc.name
end
```

Default 데이터 센터만 있는 환경에서 예제 출력:

```
Default
```

자세한 내용은 **http://www.rubydoc.info/gems/ovirt-engine-sdk/OvirtSDK4/DataCentersService:list** 을 참조하십시오.

3.3. 클러스터 나열

이 Ruby 예제에서는 클러스터를 나열합니다.

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Get the reference to the service that manages the
# collection of clusters:
cls_service = system_service.clusters_service

# Retrieve the list of clusters and for each one
# print its name:
cls = cls_service.list
cls.each do |cl|
  puts cl.name
end
```

Default 클러스터만 있는 환경에서 예제 출력:

```
Default
```

자세한 내용은 ClustersService:list 를 참조하십시오.

3.4. 논리적 네트워크 나열

이 Ruby 예제에서는 논리적 네트워크를 나열합니다.

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Get the reference to the service that manages the
# collection of networks:
nws_service = system_service.networks_service

# Retrieve the list of clusters and for each one
# print its name:
nws = nws_service.list
nws.each do |nw|
  puts nw.name
end
```

기본 관리 네트워크만 있는 환경에서 예제 출력:

```
ovirtmgmt
```

자세한 내용은 NetworksService:list 를 참조하십시오.

3.5. 호스트 나열

이 Ruby 예제에서는 호스트를 나열합니다.

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Get the reference to the service that manages the
# collection of hosts:
host_service = system_service.hosts_service

# Retrieve the list of hosts and for each one
# print its name:
host = host_service.list
host.each do |host|
  puts host.name
end
```

연결된 호스트(Atlantic)만 있는 환경에서 예제 출력:

```
Atlantic
```

자세한 내용은 HostsService:list-instance_method 를 참조하십시오.

3.6. ISO 스토리지 도메인에 ISO 파일 나열

이 Ruby 예제에서는 ISO 스토리지 도메인인 myiso 에 있는 ISO 파일을 나열합니다.

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Find the service that manages the collection of storage domains:
sds_service = system_service.storage_domains_service

# Find the ISO storage domain:
sd = sds_service.list(search: 'name=myiso').first

# Find the service that manages the ISO storage domain:
sd_service = sds_service.storage_domain_service(sd.id)

# Find the service that manages the collection of files available in the storage domain:
files_service = sd_service.files_service
```

```
# List the names of the files. Note that the name of the .iso file is contained
# in the `id` attribute.
files = files_service.list
files.each do |file|
  puts file.id
end
```

자세한 내용은 FilesService:list 를 참조하십시오.

ISO 스토리지 도메인의 이름을 모르는 경우 이 Ruby 예제에서는 모든 ISO 스토리지 도메인을 검색합니다.

```
# Find the ISO storage domain:
iso_sds = sds_service.list.select { |sd| sd.type == OvirtSDK4::StorageDomainType::ISO }
```

자세한 내용은 StorageDomainsService:list 를 참조하십시오.

3.7. NFS 스토리지 도메인 생성

이 Ruby 예제에서는 NFS 스토리지 도메인을 추가합니다.

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Get the reference to the storage domains service:
sds_service = connection.system_service.storage_domains_service

# Create a new NFS data storage domain:
sd = sds_service.add(
  OvirtSDK4::StorageDomain.new(
    name: 'mydata',
    description: 'My data',
    type: OvirtSDK4::StorageDomainType::DATA,
    host: {
      name: 'myhost'
    },
    storage: {
      type: OvirtSDK4::StorageType::NFS,
      address: 'server0.example.com',
      path: '/nfs/ovirt/40/mydata'
    }
  )
)

# Wait until the storage domain is unattached:
sd_service = sds_service.storage_domain_service(sd.id)
```

```
loop do
  sleep(5)
  sd = sd_service.get
  break if sd.status == OvirtSDK4::StorageDomainStatus::UNATTACHED
end
```

자세한 내용은 http://www.rubydoc.info/gems/ovirt-engine-sdk/OvirtSDK4/StorageDomainsService:add 을 참조하십시오.

3.8. 데이터 센터에 스토리지 도메인 연결

이 Ruby 예제에서는 기존 NFS 스토리지 도메인 mydata 를 기존 데이터 센터인 mydc 에 연결합니다. 이 예는 데이터와 ISO 스토리지 도메인을 모두 연결하는 데 사용됩니다.

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Locate the service that manages the storage domains and use it to
# search for the storage domain:
sds_service = system_service.storage_domains_service
sd = sds_service.list(search: 'name=mydata')[0]

# Locate the service that manages the data centers and use it to
# search for the data center:
dcs_service = system_service.data_centers_service
dc = dcs_service.list(search: 'name=mydc')[0]

# Locate the service that manages the data center where you want to
# attach the storage domain:
dc_service = dcs_service.data_center_service(dc.id)

# Locate the service that manages the storage domains that are attached
# to the data centers:
attached_sds_service = dc_service.storage_domains_service

# Use the "add" method of service that manages the attached storage
# domains to attach it:
attached_sds_service.add(
  OvirtSDK4::StorageDomain.new(
    id: sd.id
  )
)

# Wait until the storage domain is active:
attached_sd_service = attached_sds_service.storage_domain_service(sd.id)
loop do
  sleep(5)
  sd = attached_sd_service.get
  break if sd.status == OvirtSDK4::StorageDomainStatus::ACTIVE
end
```

자세한 내용은 http://www.rubydoc.info/gems/ovirt-engine-sdk/OvirtSDK4/StorageDomainsService:add 을 참조하십시오.

3.9. 가상 머신 생성

이 Ruby 예제에서는 가상 머신을 생성합니다. 이 예에서는 기호 및 중첩 해시가 있는 해시를 값으로 사용합니다. 또 다른 방법은 해당 오브젝트의 생성자를 직접 사용하는 것입니다. 자세한 내용은 Creating a Virtual Machine Instance with Attributes 를 참조하십시오.

```
# Get the reference to the "vms" service:
vms_service = connection.system_service.vms_service

# Use the "add" method to create a new virtual machine:
vms_service.add(
  OvirtSDK4::Vm.new(
    name: 'myvm',
    cluster: {
      name: 'mycluster'
    },
    template: {
      name: 'Blank'
    }
  )
)
```

가상 머신을 생성한 후에는 가상 시스템의 상태를 폴링 하여 모든 디스크가 생성되었는지 확인하는 것이 좋습니다.

자세한 내용은 VmsService:add 를 참조하십시오.

3.10. VNIC 프로필 생성

이 Ruby 예제에서는 vNIC 프로필을 생성합니다.

```
# Find the root of the tree of services:
system_service = connection.system_service

# Find the network where you want to add the profile. There may be multiple
# networks with the same name (in different data centers, for example).
# Therefore, you must look up a specific network by name, in a specific data center.
dcs_service = system_service.data_centers_service
dc = dcs_service.list(search: 'name=mydc').first
networks = connection.follow_link(dc.networks)
network = networks.detect { |n| n.name == 'mynetwork' }
```

```
# Create the vNIC profile, with passthrough and port mirroring disabled:
profiles_service = system_service.vnic_profiles_service
profiles_service.add(
  OvirtSDK4::VnicProfile.new(
    name: 'myprofile',
    pass_through: {
      mode: OvirtSDK4::VnicPassThroughMode::DISABLED,
    },
    port_mirroring: false,
    network: {
      id: network.id
    }
  )
)
```

자세한 내용은 VnicProfilesService:add 를 참조하십시오.

3.11. VNIC 생성

새로 생성된 가상 시스템에 네트워크 액세스 권한이 있는지 확인하려면 **vNIC**를 생성하고 연결해야 합니다.

이 **Ruby** 예제에서는 **vNIC**를 생성하고 기존 가상 머신인 **myvm** 에 연결합니다.

```
# Find the root of the tree of services:
system_service = connection.system_service

# Find the virtual machine:
vms_service = system_service.vms_service
vm = vms_service.list(search: 'name=myvm').first

# In order to specify the network that the new NIC will be connected to, you must
# specify the identifier of the vNIC profile. However, there may be multiple
# profiles with the same name (for different data centers, for example), so first
# you must find the networks that are available in the cluster that the
# virtual machine belongs to.
cluster = connection.follow_link(vm.cluster)
networks = connection.follow_link(cluster.networks)
network_ids = networks.map(&:id)

# Now that you know what networks are available in the cluster, you can select a
# vNIC profile that corresponds to one of those networks, and has the
# name that you want to use. The system automatically creates a vNIC
# profile for each network, with the same name as the network.
profiles_service = system_service.vnic_profiles_service
profiles = profiles_service.list
profile = profiles.detect { |p| network_ids.include?(p.network.id) && p.name == 'myprofile' }
```

```
# Locate the service that manages the network interface cards collection of the
# virtual machine:
nics_service = vms_service.vm_service(vm.id).nics_service

# Add the new network interface card:
nics_service.add(
  OvirtSDK4::Nic.new(
    name: 'mynic',
    description: 'My network interface card',
    vnic_profile: {
      id: profile.id
    }
  )
)
```

자세한 내용은 VmsService:add 를 참조하십시오.

3.12. 가상 디스크 생성

새로 생성된 가상 머신이 영구 스토리지에 액세스할 수 있도록 하려면 디스크를 생성하고 연결해야 합니다.

이 Ruby 예제에서는 가상 스토리지 디스크를 생성하고 가상 머신에 연결합니다.

```
# Locate the virtual machines service and use it to find the virtual
# machine:
vms_service = connection.system_service.vms_service
vm = vms_service.list(search: 'name=myvm')[0]

# Locate the service that manages the disk attachments of the virtual
# machine:
disk_attachments_service = vms_service.vm_service(vm.id).disk_attachments_service

# Use the "add" method of the disk attachments service to add the disk.
# Note that the size of the disk, the `provisioned_size` attribute, is
# specified in bytes, so to create a disk of 10 GiB the value should
# be 10 * 2^30.
disk_attachment = disk_attachments_service.add(
  OvirtSDK4::DiskAttachment.new(
    disk: {
      name: 'mydisk',
      description: 'My disk',
      format: OvirtSDK4::DiskFormat::COW,
      provisioned_size: 10 * 2**30,
      storage_domains: [{
        name: 'mydata'
      }]
    },
    interface: OvirtSDK4::DiskInterface::VIRTIO,
```

```
    bootable: false,
    active: true
  )
)

# Wait until the disk status is OK:
disks_service = connection.system_service.disks_service
disk_service = disks_service.disk_service(disk_attachment.disk.id)
loop do
  sleep(5)
  disk = disk_service.get
  break if disk.status == OvirtSDK4::DiskStatus::OK
end
```

자세한 내용은 DiskAttachmentsService:add 를 참조하십시오.

3.13. 가상 머신에 ISO 이미지 연결

이 Ruby 예제에서는 가상 머신에 CD-ROM을 연결하고 게스트 운영 체제를 설치하기 위해 ISO 이미지로 변경합니다.

```
# Get the reference to the "vms" service:
vms_service = connection.system_service.vms_service

# Find the virtual machine:
vm = vms_service.list(search: 'name=myvm')[0]

# Locate the service that manages the virtual machine:
vm_service = vms_service.vm_service(vm.id)

# Locate the service that manages the CDROM devices of the VM:
cdroms_service = vm_service.cdroms_service

# List the first CDROM device:
cdrom = cdroms_service.list[0]

# Locate the service that manages the CDROM device you just found:
cdrom_service = cdroms_service.cdrom_service(cdrom.id)

# Change the CD of the VM to 'my_iso_file.iso'. By default this
# operation permanently changes the disk that is visible to the
# virtual machine after the next boot, but it does not have any effect
# on the currently running virtual machine. If you want to change the
# disk that is visible to the current running virtual machine, change
# the `current` parameter's value to `true`.
cdrom_service.update(
  OvirtSDK4::Cdrom.new(
    file: {
      id: 'CentOS-7-x86_64-DVD-1511.iso'
    }
```

```
  ),
  current: false
)
```

자세한 내용은 VmService:cdroms_service 를 참조하십시오.

3.14. 가상 디스크 분리

이 Ruby 예제에서는 가상 머신에서 디스크를 분리합니다.

```
# Find the virtual machine:
vm = vms_service.list(search: 'name=myvm').first

# Detach the first disk from the virtual machine:
vm_service = vms_service.vm_service(vm.id)
attachments_service = vm_service.disk_attachments_service
attachment = attachments_service.list.first

# Remove the attachment. The default behavior is that the disk is detached
# from the virtual machine, but not deleted from the system. If you wish to
# delete the disk, change the `detach_only` parameter to `false`.
attachment.remove(detach_only: true)
```

자세한 내용은 VmService:disk_attachments_service 를 참조하십시오.

3.15. 가상 머신 시작

이 Ruby 예제에서는 가상 머신을 시작합니다.

```
# Get the reference to the "vms" service:
vms_service = connection.system_service.vms_service

# Find the virtual machine:
vm = vms_service.list(search: 'name=myvm')[0]

# Locate the service that manages the virtual machine, as that is where
# the action methods are defined:
vm_service = vms_service.vm_service(vm.id)

# Call the "start" method of the service to start it:
vm_service.start

# Wait until the virtual machine status is UP:
loop do
  sleep(5)
```

```
  vm = vm_service.get
  break if vm.status == OvirtSDK4::VmStatus::UP
end
```

자세한 내용은 VmService:start 를 참조하십시오.

3.16. 특정 부팅 장치 및 부팅 순서를 사용하여 가상 머신 시작

이 Ruby 예제에서는 부팅 장치 및 부팅 순서를 지정하는 가상 머신을 시작합니다.

```
# Find the root of the tree of services:
system_service = connection.system_service

# Find the virtual machine:
vms_service = system_service.vms_service
vm = vms_service.list(search: 'name=myvm').first

# Find the service that manages the virtual machine:
vm_service = vms_service.vm_service(vm.id)

# Start the virtual machine explicitly indicating the boot devices and order:
vm_service.start(
  vm: {
    os: {
      boot: {
        devices: [
          OvirtSDK4::BootDevice::NETWORK,
          OvirtSDK4::BootDevice::CDROM
        ]
      }
    }
  }
)

# Wait until the virtual machine is up:
loop do
  sleep(5)
  vm = vm_service.get
  break if vm.status == OvirtSDK4::VmStatus::UP
end
```

자세한 내용은 BootDevice 를 참조하십시오.

3.17. CLOUD-INIT를 사용하여 가상 머신 시작

이 Ruby 예제에서는 Cloud-Init 도구를 사용하여 가상 머신을 시작하여 루트 암호 및 네트워크 구성을 설정합니다.

```
# Find the virtual machine:
vms_service = connection.system_service.vms_service
vm = vms_service.list(search: 'name=myvm')[0]

# Find the service that manages the virtual machine:
vm_service = vms_service.vm_service(vm.id)

# Create a cloud-init script to execute in the
# deployed virtual machine. The script must be correctly
# formatted and indented because it uses YAML.
my_script = ''
write_files:
 - content: |
     Hello, world!
   path: /tmp/greeting.txt
   permissions: '0644'
''

# Start the virtual machine, enabling cloud-init and providing the
# password for the `root` user and the network configuration:
vm_service.start(
 use_cloud_init: true,
 vm: {
  initialization: {
   user_name: 'root',
   root_password: 'redhat123',
   host_name: 'myvm.example.com',
   nic_configurations: [
    {
     name: 'eth0',
     on_boot: true,
     boot_protocol: OvirtSDK4::BootProtocol::STATIC,
     ip: {
      version: OvirtSDK4::IpVersion::V4,
      address: '192.168.0.100',
      netmask: '255.255.255.0',
      gateway: '192.168.0.1'
     }
    }
   ],
   dns_servers: '192.168.0.1 192.168.0.2 192.168.0.3',
   dns_search: 'example.com',
   custom_script: my_script
  }
 }
)
```

자세한 내용은 VmService:start 를 참조하십시오.

3.18. 시스템 이벤트 확인

이 Ruby 예제에서는 기록된 시스템 이벤트를 검색합니다.

```
# In order to ensure that no events are lost, it is recommended to write
# the index of the last processed event, in persistent storage.
# Here, it is stored in a file called `index.txt`. In a production environment,
# it will likely be stored in a database.
INDEX_TXT = 'index.txt'.freeze

def write_index(index)
  File.open(INDEX_TXT, 'w') { |f| f.write(index.to_s) }
end

def read_index
  return File.read(INDEX_TXT).to_i if File.exist?(INDEX_TXT)
  nil
end

# This is the function that is called to process the events. It prints
# the identifier and description of each event.
def process_event(event)
  puts("#{event.id} - #{event.description}")
end

# Find the root of the tree of services:
system_service = connection.system_service

# Find the service that manages the collection of events:
events_service = system_service.events_service

# If no index is stored yet, retrieve the last event and start with it.
# Events are ordered by index, in ascending order. `max=1` retrieves only one event,
# the last event.
unless read_index
  events = events_service.list(max: 1)
  unless events.empty?
    first = events.first
    process_event(first)
    write_index(first.id.to_i)
  end
end

# This loop retrieves the events, always starting from the last index. It waits
# before repeating. The `from` parameter specifies that you want to retrieve
# events that are newer than the last index that was processed. Note: the `max`
# parameter is not used, so that all pending events will be retrieved.
loop do
  sleep(5)
  events = events_service.list(from: read_index)
  events.each do |event|
    process_event(event)
    write_index(event.id.to_i)
  end
end
```

자세한 내용은 **EventsService:list** 를 참조하십시오.

부록 A. 법적 통지