Red Hat Virtualization 4.4

Python SDK 가이드

Red Hat Virtualization Python SDK 사용

Last Updated: 2022-06-26

Red Hat Virtualization 4.4 Python SDK 가이드

Red Hat Virtualization Python 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 Python 소프트웨어 개발 키트 버전 4를 설치하고 사용하는 방법을 설명합니다.

차례

1장. 개요 **3**
1.1. 사전 요구 사항 3
1.2. PYTHON 소프트웨어 개발 키트 설치 4

2장. 소프트웨어 개발 키트 사용 **5**
2.1. 패키지 5
2.2. 서버에 연결CONNECT TO THE SERVER 5
2.3. 유형 사용 6
2.4. 링크 사용 7
2.5. 서비스 검색 8
2.6. 서비스 사용 9
2.6.1. get Methods 사용 9
2.6.2. 목록 방법 사용 10
2.6.3. 추가 방법 사용 11
2.6.4. 업데이트 방법 사용 12
2.6.5. 제거 방법 사용 14
2.6.6. 기타 작업 방법 사용 15
2.7. 추가 리소스 15
2.7.1. 모듈에 대한 문서 생성 15

3장. PYTHON 예 **17**
3.1. 개요 17
3.2. 버전 4에서 RED HAT VIRTUALIZATION MANAGER 연결 18
3.3. 데이터 센터 나열 20
3.4. 클러스터 나열 21
3.5. 호스트 나열 22
3.6. 논리적 네트워크 나열 23
3.7. 가상 머신 및 총 디스크 크기 나열 23
3.8. NFS 데이터 스토리지 생성 24
3.9. NFS ISO 스토리지 생성 26
3.10. 데이터 센터에 스토리지 도메인 연결 28
3.11. 스토리지 도메인 활성화 30
3.12. ISO 스토리지 도메인의 파일 나열 31
3.13. 가상 머신 생성 32
3.14. 가상 NIC 생성 33
3.15. 가상 머신 디스크 생성 35
3.16. 가상 머신에 ISO 이미지 연결 37
3.17. 디스크 분리 39
3.18. 가상 머신 시작 40
3.19. OVERRIDDEN 매개변수를 사용하여 가상 머신 시작 41
3.20. CLOUD-INIT를 사용하여 가상 머신 시작 43
3.21. 시스템 이벤트 확인 44

부록 A. 법률 알림 **46**

1장. 개요

Python 소프트웨어 개발 키트의 버전 4는 Python 기반 프로젝트에서 Red Hat Virtualization Manager와 상호 작용할 수 있는 클래스 컬렉션입니다. 이러한 클래스를 다운로드하여 프로젝트에 추가하면 관리 작업의 고급 자동화를 위해 다양한 기능에 액세스할 수 있습니다.

참고

SDK 버전 3은 더 이상 지원되지 않습니다. 자세한 내용은 이 가이드의 RHV 4.3 버전 을 참조하십시오.

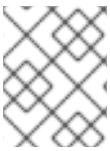
Python 3.7 및 async

Python 3.7 이상 버전에서 **async** 는 reserved 키워드입니다. **async =True** 가 오류를 유발하기 때문에 이전에 지원하는 서비스 방법에서는 async 매개 변수를 사용할 수 없습니다.

```
dc = dc_service.update(
    types.DataCenter(
        description='Updated description',
    ),
    async=True,
)
```

해결책은 매개 변수 (**async_**)에 밑줄을 추가하는 것입니다.

```
dc = dc_service.update(
    types.DataCenter(
        description='Updated description',
    ),
    async_=True,
)
```



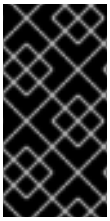
참고

이 제한은 Python 3.7 이상에만 적용됩니다. 이전 버전의 Python에는 이 수정이 필요하지 않습니다.

1.1. 사전 요구 사항

Python 소프트웨어 개발 키트를 설치하려면 다음이 필요합니다.

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



중요

소프트웨어 개발 키트는 Red Hat Virtualization REST API를 위한 인터페이스입니다. Red Hat Virtualization 환경의 버전에 해당하는 소프트웨어 개발 키트 버전을 사용합니다. 예를 들어 Red Hat Virtualization 4.3을 사용하는 경우 V4 Python 소프트웨어 개발 키트를 사용하십시오.

1.2. PYTHON 소프트웨어 개발 키트 설치

Python 소프트웨어 개발 키트를 설치하려면 다음을 수행합니다.

1. 하드웨어 플랫폼에 적합한 리포지토리를 활성화합니다. 예를 들어 x86-64 하드웨어의 경우 다음을 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. 필수 패키지를 설치합니다.

```
# dnf install python3-ovirt-engine-sdk4
```

Python 소프트웨어 개발 키트는 Python 3 site-packages 디렉터리에 설치되고 관련 문서 및 예제는 **/usr/share/doc/python3-ovirt-engine-sdk4**.

2장. 소프트웨어 개발 키트 사용

이 섹션에서는 버전 4용 소프트웨어 개발 키트를 사용하는 방법을 설명합니다.

2.1. 패키지

다음 모듈은 Python SDK에서 가장 자주 사용됩니다.

ovirtsdk4

이것이 최상위 모듈입니다. 가장 중요한 요소는 서버에 연결하고 서비스 트리의 루트에 대한 참조를 얻는 메커니즘인 **Connection** 클래스입니다.
Error 클래스는 오류를 보고해야 할 때 SDK가 발생할 기본 예외 클래스입니다.

특정 종류의 오류의 경우 기본 오류 클래스를 확장하는 특정 오류 클래스가 있습니다.

- **AuthError** - 인증 또는 권한 부여가 실패할 때 조정됩니다.
- **ConnectionError** - 서버 이름을 확인할 수 없거나 서버에 연결할 수 없는 경우입니다.
- **NotFoundError** - 요청된 개체가 없을 때 분류됩니다.
- **TimeoutError** - 작업이 시간 초과될 때 지연됩니다.

ovirtsdk4.types

이 모듈에는 API에서 사용되는 유형을 구현하는 클래스가 포함되어 있습니다. 예를 들어 **ovirtsdk4.types.Vm** 클래스는 가상 머신 유형의 구현입니다. 이러한 클래스는 데이터 컨테이너이며 논리를 포함하지 않습니다.
이러한 클래스의 인스턴스는 매개 변수로 사용되며 서비스 메서드의 값을 반환합니다. 기본 표현 또는 기본 표현으로의 변환은 SDK에 의해 투명하게 처리됩니다.

ovirtsdk4.services

이 모듈에는 API에서 지원하는 서비스를 구현하는 클래스가 포함되어 있습니다. 예를 들어 **ovirtsdk4.services.VmsService** 클래스는 시스템의 가상 머신 컬렉션을 관리하는 서비스 구현입니다.
이러한 클래스의 인스턴스는 서비스가 있는 경우 SDK에 의해 자동으로 생성됩니다. 예를 들어 다음을 수행할 때 SDK에 의해 **VmsService** 클래스의 새 인스턴스가 자동으로 생성됩니다.

```
vms_service = connection.system_service().vms_service()
```

생성자의 매개 변수로, 일반적으로 서비스 로케이터 및 서비스 메서드를 제외한 모든 메서드가 변경될 수 있으므로 이러한 클래스의 인스턴스를 수동으로 생성하지 않는 것이 가장 좋습니다.

ovirtsdk4.http,**ovirtsdk4.readers**, **ovirtsdk4.writers** 와 같은 다른 모듈이 있습니다. HTTP 통신 및 XML 구문 분석 및 렌더링을 구현하는 데 사용됩니다. 나중에 변경될 수 있는 내부 구현 세부 정보이기 때문에 이를 사용하지 마십시오. 이전 버전과의 호환성은 보장되지 않습니다.

2.2. 서버에 연결CONNECT TO THE SERVER

서버에 연결하려면 **Connection** 클래스가 포함된 **ovirtsdk4** 모듈을 가져옵니다. 이는 SDK의 진입점이며 API의 서비스 트리의 루트에 대한 액세스를 제공합니다.

```
import ovirtsdk4 as sdk
```

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

연결에는 서버에 대한 HTTP 연결 풀 및 인증 토큰을 포함하여 중요한 리소스가 있습니다. 이러한 리소스를 더 이상 사용하지 않을 때 리소스를 해제하는 것이 매우 중요합니다.

```
connection.close()
```

연결이 닫히면 다시 사용할 수 없습니다.

TLS로 보호되는 서버에 연결할 때 **ca.pem** 파일이 필요합니다. 일반적인 설치에서는 Manager 시스템의 **/etc/pki/ovirt-engine/** 에 있습니다. **ca_file** 을 지정하지 않으면 시스템 전체 CA 인증서 저장소가 사용됩니다. **ca.pem** 파일을 가져오는 방법에 대한 자세한 내용은 *REST API 가이드* 를 참조하십시오.

연결에 실패하면 SDK는 세부 정보가 포함된 **ovirtsdk4.Error** 예외를 발생시킵니다.

2.3. 유형 사용

ovirtsdk4.types 모듈의 클래스는 순수 데이터 컨테이너입니다. 논리 또는 작업이 없습니다. 필요에 따라 형식 인스턴스를 만들고 수정할 수 있습니다.

아래에 설명된 서비스 메서드 중 하나에 대한 호출과 함께 변경 사항을 명시적으로 전달하지 않는 한 인스턴스를 생성하거나 수정해도 서버 쪽에는 영향을 미치지 않습니다. 서버 측의 변경 사항은 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 단순화하기 위한 것입니다. 이 예제에서는 가상 머신의 인스턴스를 생성하여 클러스터 이름, 템플릿 및 메모리를 바이트 단위로 지정합니다.

```
from ovirtsdk4 import types

vm = types.Vm(
    name='vm1',
    cluster=types.Cluster(
        name='Default'
    ),
    template=types.Template(
        name='mytemplate'
    ),
    memory=1073741824
)
```

이러한 방식으로 생성자를 사용하는 것이 권장되지만 필수는 아닙니다. 생성자 호출에서 인수 없이 인스턴스를 생성하고, setters를 사용하여 단계별로 개체 단계를 채우거나 두 방법의 조합을 사용하여 오브젝트 단계를 채울 수도 있습니다.

```
vm = types.Vm()
vm.name = 'vm1'
vm.cluster = types.Cluster(name='Default')
```

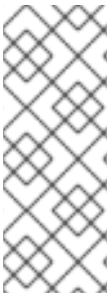
```
vm.template = types.Template(name='mytemplate')
vm.memory=1073741824
```

API 사양에서 오브젝트 목록으로 정의된 속성은 Python 목록으로 구현됩니다. 예를 들어 **Vm** 유형의 **custom_properties** 속성은 **CustomProperty** 유형의 개체 목록으로 정의됩니다. SDK에서 속성이 사용되는 경우 Python 목록이 됩니다.

```
vm = types.Vm(
    name='vm1',
    custom_properties=[
        types.CustomProperty(...),
        types.CustomProperty(...),
        ...
    ]
)
```

API에서 열거된 값으로 정의된 속성은 Python 3의 원시 지원 및 Python 2.7의 free34 패키지 사용 하 여 Python에서num으로 구현 됩니다.Attributes that are defined as enumerated values in API are implemented as enumerated values in Python, using the native support for enumerate **s** in Python 3 and the free **34** package in Python 2.7. **이 예에서 Vm 유형의 status 속성은 VmStatus num을 사용하여 정의됩니다.**

```
if vm.status == types.VmStatus.DOWN:
    ...
elif vm.status == types.VmStatus.IMAGE_LOCKED:
    ....
```



참고

API 사양에서는 XML 및 JSON에 사용되는 형식 값이 소문자로 표시됩니다.In the API specification, the values of enumerate types appear in lower case, because that is what is used for XML and JSON. 그러나 Python 규칙은num 값을 대문자로 지정하는 것입니다.

해당 속성을 사용하여 형식 인스턴스의 특성을 읽습니다.Reading the attributes of instances of types is done using the corresponding properties:

```
print("vm.name: %s" % vm.name)
print("vm.memory: %s" % vm.memory)
for custom_property in vm.custom_properties:
    ...
```

2.4. 링크 사용

유형의 일부 속성은 API에서 링크로 정의합니다. 이 규칙은 해당 오브젝트의 표현을 검색할 때 값이 일반적으로 채워지지 않음을 나타냅니다. 대신 링크가 반환됩니다. 예를 들어 가상 머신을 검색할 때 서버의

XML 응답에는 **< link>** 특성이 포함됩니다.

```
<vm id="123" href="/ovirt-engine/api/vms/123">
  <name>vm1</name>
  <link rel="diskattachments" href="/ovirt-engine/api/vms/123/diskattachments/>
  ...
</vm>
```

vm.diskattachments 에 대한 링크는 실제 디스크 첨부 파일이 포함되어 있지 않습니다. 데이터를 가져오기 위해 **Connection** 클래스는 **href XML** 속성 값을 사용하여 실제 데이터를 검색하는 **follow_link** 메서드를 제공합니다. 예를 들어 가상 머신의 디스크 세부 정보를 검색하려면 디스크 첨부 파일에 대한 링크를 수행한 다음 각 디스크에 대한 링크를 따릅니다.

```
# Retrieve the virtual machine:
vm = vm_service.get()

# Follow the link to the disk attachments, and then to the disks:
attachments = connection.follow_link(vm.disk_attachments)
for attachment in attachments:
  disk = connection.follow_link(attachment.disk)
  print("disk.alias: " % disk.alias)
```

2.5. 서비스 검색

API는 서버 **URL** 공간 내의 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.

SDK에서 해당 서비스 트리의 루트는 시스템 서비스에 의해 구현됩니다. 연결의 **system_service** 메서드를 호출할 수 있습니다.

```
system_service = connection.system_service()
```

이 시스템 서비스에 대한 참조가 있는 경우 이를 사용하여 이전 서비스의 ***_service** 메서드를 호출하여 다른 서비스에 대한 참조를 가져올 수 있습니다. 예를 들어 시스템의 가상 머신 컬렉션을 관리하는 서비스에 대한 참조를 얻으려면 **vms_service** 서비스 **locator**를 사용합니다.

```
vms_service = system_service.vms_service()
```

식별자가 **123** 인 가상 머신을 관리하는 서비스에 대한 참조를 얻으려면 가상 머신의 컬렉션을 관리하는 서비스의 **vm_service** 서비스 로케이터를 사용합니다. 가상 머신의 식별자를 매개변수로 사용합니다.

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

중요

서비스 로케이터를 호출해도 서버에 요청을 보내지 않습니다. 반환되는 **Python** 오브젝트는 데이터를 포함하지 않는 순수 서비스입니다. 예를 들어 이 예제에서 호출한 **vm_service Python** 오브젝트는 가상 머신의 표현이 아닙니다. 해당 가상 머신을 검색, 업데이트, 삭제, 시작 및 중지하는 데 사용되는 서비스입니다.

2.6. 서비스 사용

서비스를 찾은 후에는 서버에 요청을 보내고 실제 작업을 수행하는 서비스 메서드를 호출할 수 있습니다.

단일 오브젝트를 관리하는 서비스는 일반적으로 **get,update** 및 **remove** 메서드를 지원합니다.

오브젝트 컬렉션을 관리하는 서비스는 일반적으로 목록 및 추가 방법을 지원합니다.

두 종류의 서비스, 특히 단일 개체를 관리하는 서비스는 모두 추가 작업 메서드를 지원할 수 있습니다.

2.6.1. get Methods 사용

이러한 서비스 메서드는 단일 개체의 표현을 검색하는 데 사용됩니다. 다음 예제에서는 식별자가 **123**인 가상 머신의 표현을 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()
```

응답은 해당 유형의 인스턴스이며 이 경우 **Python** 클래스 **ovirtsdk4.types.Vm.**

일부 서비스의 **get** 메서드는 개체의 표현을 검색하는 방법 또는 하나 이상의 경우 검색할 표현 방법을 제어하는 추가 매개 변수를 지원합니다. 예를 들어 가상 시스템의 현재 상태 또는 다음에 다른 상태가 시작될 때 해당 상태를 검색할 수 있습니다. 가상 머신을 관리하는 서비스의 **get** 메서드는 **next_run** 부울 매개 변수를 지원합니다.

```
# 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 SDK의 참조 문서를 참조하십시오.

어떤 이유로든 오브젝트를 검색할 수 없는 경우 SDK는 실패 세부 정보와 함께 ovirtsdk4.Error 예외가 발생합니다. 여기에는 개체가 실제로 존재하지 않는 상황이 포함됩니다. get service 메서드를 호출할 때 예외가 발생합니다. 해당 호출이 서버에 요청을 보내지 않기 때문에 오브젝트가 없는 경우에도 서비스 로케이터 메서드 호출이 실패하지 않습니다. 예:

```
# Call the service that manages a non-existent virtual machine.
# This call will succeed.
vm_service = vms_service.vm_service('junk')

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

2.6.2. 목록 방법 사용

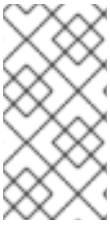
이러한 서비스 메서드는 컬렉션의 개체의 표현을 검색합니다.These service methods retrieve the representations of the objects of a collection. 이 예에서는 시스템의 가상 머신의 전체 컬렉션을 검색합니다.

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

# List the virtual machines in the collection
vms = vms_service.list()
```

그 결과 해당 유형의 인스턴스를 포함하는 Python 목록이 생성됩니다. 예를 들어 이 경우 결과는 ovirtsdk4.types.Vm 클래스의 인스턴스 목록이 됩니다.

일부 서비스의 목록 방법은 추가 매개 변수를 지원합니다. 예를 들어 거의 모든 최상위 컬렉션에서는 검색 매개 변수를 지원하여 결과 또는 max 매개 변수를 필터링하여 서버에서 반환된 결과 수를 제한합니다. 이 예에서는 상위 10개의 결과인 my 부터 시작하는 가상 머신의 이름을 검색합니다.

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



참고

일부 목록 방법이 이러한 매개 변수를 지원하는 것은 아닙니다. 일부 목록 방법은 다른 매개 변수를 지원합니다. 자세한 내용은 SDK의 참조 문서를 참조하십시오.

어떤 이유로든 반환된 결과 목록이 비어 있으면 반환된 값이 빈 목록이 됩니다. 이는 절대로 None 이 될 수 없습니다.

결과를 검색하는 동안 오류가 발생하면 SDK에서 오류 세부 정보가 포함된 ovirtsdk4.Error 예외가 발생합니다.

2.6.3. 추가 방법 사용

이러한 서비스 메서드는 컬렉션에 새 요소를 추가합니다. 추가할 오브젝트를 설명하는 관련 유형의 인스턴스를 수신하고, 추가할 요청을 보내고, 추가된 오브젝트를 설명하는 유형의 인스턴스를 반환합니다.

이 예제에서는 vm1 이라는 새 가상 머신을 추가합니다.

```
from ovirtsdk4 import types

# Add the virtual machine:
vm = vms_service.add(
    vm=types.Vm(
        name='vm1',
        cluster=types.Cluster(
            name='Default'
        ),
        template=types.Template(
            name='mytemplate'
        )
    )
)
```

어떤 이유로든 오브젝트를 생성할 수 없는 경우 SDK는 실패 세부 정보가 포함된 ovirtsdk4.Error 예외를 발생시킵니다. 아무 것도 반환하지 않습니다.

중요

이 **add** 메서드에서 반환된 **Python** 오브젝트는 관련 유형의 인스턴스입니다. 이는 서비스가 아니라 데이터 컨테이너입니다. 이 특정 예제에서 반환된 오브젝트는 **ovirtsdk4.types.Vm** 클래스의 인스턴스입니다. 가상 머신을 생성한 후 가상 머신 검색 또는 시작과 같은 작업을 수행해야 하는 경우 먼저 이를 관리하는 서비스를 찾고 해당 서비스 로케이터를 호출해야 합니다.

```
# 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()
```

개체는 비동기적으로 생성됩니다.**Objects are created asynchronously.** 새 가상 머신을 생성할 때 추가 방법은 가상 머신을 완전히 생성하고 사용할 준비가 되기 전에 응답을 반환합니다. 오브젝트 상태를 폴링하여 완전히 생성되었는지 확인하는 것이 좋습니다. 가상 머신의 경우 해당 상태가 **DOWN** 이 될 때까지 확인해야 합니다.

```
# 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 it is
# completely created:
while True:
    time.sleep(5)
    vm = vm_service.get()
    if vm.status == types.VmStatus.DOWN:
        break
```

반복문을 사용하여 **get** 메서드와 함께 오브젝트 상태를 검색하여 **status** 속성이 업데이트되었는지 확인합니다.

2.6.4. 업데이트 방법 사용

이러한 서비스 방법은 기존 오브젝트를 업데이트합니다. 수행할 업데이트를 설명하고, 업데이트를 위해 요청을 보내고, 업데이트된 오브젝트를 설명하는 유형의 인스턴스를 반환합니다.

이 예에서는 **vm1** 에서 **newvm** 으로 가상 머신의 이름을 업데이트합니다.

```
from ovirtsdk4 import types

# Find the virtual machine, and then the service that
# manages it:
vm = vms_service.list(search='name=vm1')[0]
vm_service = vm_service.vm_service(vm.id)

# Update the name:
updated_vm = vm_service.update(
   vm=types.Vm(
      name='newvm'
   )
)
```

업데이트를 수행할 때 개체의 전체 표현은 보내지 않도록 합니다. 업데이트하려는 속성만 보냅니다. 이 작업을 수행하지 마십시오.

```
# Retrieve the complete representation:
vm = vm_service.get()

# Update the representation, in memory, without sending a request
# to the server:
vm.name = 'newvm'

# Send the update. Do *not* do this.
vms_service.update(vm)
```

완전한 표현을 전송하면 두 가지 문제가 발생합니다.

- 서버에 필요한 것보다 더 많은 정보를 보내고 있으므로 리소스를 낭비하고 있습니다.
- 서버는 변경하려는 경우에도 오브젝트의 모든 속성을 업데이트하려고 시도합니다. 이로 인해 서버 측에서 버그가 발생할 수 있습니다.

일부 서비스의 업데이트 방법은 업데이트 방법 또는 업데이트 방법을 제어하는 추가 매개 변수를 지원합니다. 예를 들어 다음에 가상 머신이 시작될 때 가상 머신의 현재 상태 또는 사용 상태를 업데이트할 수 있습니다. 가상 머신을 관리하는 서비스의 업데이트 방법은 **next_run** 부울 매개 변수를 지원합니다.

```
# Update the memory of the virtual machine to 1 GiB,
```

```
# not during the current run, but after next boot:
vm = vm_service.update(
   vm=types.Vm(
      memory=1073741824
   ),
   next_run=True
)
```

어떠한 이유로든 업데이트를 수행할 수 없는 경우 SDK는 오류에 대한 세부 정보가 포함된 ovirtsdk4.Error 예외를 발생시킵니다. 아무 것도 반환하지 않습니다.

이 업데이트 메서드에서 반환된 Python 오브젝트는 관련 유형의 인스턴스입니다. 이는 서비스가 아니라 데이터 컨테이너입니다. 이 특정 예제에서 반환된 오브젝트는 ovirtsdk4.types.Vm 클래스의 인스턴스입니다.

2.6.5. 제거 방법 사용

이러한 서비스 메서드는 기존 오브젝트를 제거합니다. 일반적으로는 단일 오브젝트를 관리하는 서비스 방식이므로 매개 변수를 사용하지 않습니다. 따라서 이 서비스는 제거할 오브젝트를 이미 알고 있습니다.

이 예제에서는 식별자가 123 인 가상 머신을 제거합니다.

```
# Find the virtual machine by name:
vm = vms_service.list(search='name=123')[0]

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

# Remove the virtual machine:
vm_service.remove()
```

일부 서비스의 remove 메서드는 제거 방법 또는 항목을 제어하는 추가 매개 변수를 지원합니다. 예를 들어 detach_only 부울 매개 변수를 사용하여 디스크를 유지하면서 가상 머신을 제거할 수 있습니다.

```
# Remove the virtual machine while preserving the disks:
vm_service.remove(detach_only=True)
```

Remove 메서드는 개체가 성공적으로 제거되면 None 을 반환합니다.The remove method returns None if the object is removed successfully. 삭제된 개체를 반환하지 않습니다. 어떤 이유로든 오브젝트를 제거할 수 없는 경우 SDK는 실패 세부 정보가 포함된 ovirtsdk4.Error 예외를 발생시킵니다.

2.6.6. 기타 작업 방법 사용

가상 머신 중지 및 시작과 같은 기타 작업을 수행하는 다른 서비스 방법이 있습니다.

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

이러한 방법 중 다수에는 작업을 수정하는 매개변수가 포함되어 있습니다. 예를 들어 가상 머신을 시작하는 방법은 **cloud-init** 를 사용하여 시작하려면 **use_cloud_init** 매개변수를 지원합니다.

```
# Start the virtual machine:
vm_service.start(cloud_init=True)
```

대부분의 작업 메서드가 성공하면 **None** 을 반환하고, 실패한 경우 **ovirtsdk4.Error** 를 발생시킵니다. 몇 가지 작업 메서드에서는 값을 반환합니다. 예를 들어 스토리지 도메인을 관리하는 서비스에는 스토리지 도메인이 이미 데이터 센터에 연결되어 있는지 확인하고 부울 값을 반환하는 **is_Attached** 작업 메서드가 있습니다.

```
# Check if the storage domain is attached to a data center:
sds_service = system_service.storage_domains_service()
sd_service = sds_service.storage_domain_service('123')
if sd_service.is_attached():
    ...
```

SDK의 참조 문서를 확인하여 각 서비스에서 지원하는 작업 방법, 수행하는 매개변수 및 반환되는 값을 확인합니다.

2.7. 추가 리소스

자세한 정보와 예제는 다음 리소스를 참조하십시오.

- **V4 REST API** 가이드
- **Python SDK** 참조 문서
- **Python SDK** 예

2.7.1. 모듈에 대한 문서 생성

다음 모듈에 pydoc 을 사용하여 문서를 생성할 수 있습니다.

- **ovirtsdk.api**
- **ovirtsdk.infrastructure.brokers**
- **ovirtsdk.infrastructure.errors**

문서는 **ovirt-engine-sdk-python** 패키지에서 제공합니다. **Manager** 머신에서 다음 명령을 실행하여 다음 문서의 최신 버전을 확인합니다.

```
$ pydoc [MODULE]
```

3장. PYTHON 예

3.1. 개요

이 섹션에서는 Python SDK를 사용하여 기본 Red Hat Virtualization 환경에서 가상 머신을 생성하는 단계를 설명하는 예를 제공합니다.

이 예제에서는 ovirt-engine-sdk-python 패키지에서 제공하는 ovirtsdk Python 라이브러리를 사용합니다. 이 패키지는 Red Hat Subscription Manager에서 Red Hat Virtualization 서브스크립션 풀에 연결된 시스템에서 사용할 수 있습니다. 소프트웨어를 다운로드하려면 시스템 구독에 대한 자세한 내용은 소프트웨어 개발 키트 설치를 참조하십시오.

또한 다음이 필요합니다.

- Red Hat Virtualization Manager의 네트워크 설치.
- Red Hat Virtualization Host를 연결 및 구성했습니다.
- 가상 시스템에 설치할 운영 체제를 포함하는 ISO 이미지 파일.
- Red Hat Virtualization 환경을 구성하는 논리 오브젝트와 물리적 개체를 모두 이해하고 있습니다.
- Python 프로그래밍 언어에 대한 이해

예를 들면 인증 세부 사항(사용자 이름의 경우admin@internal, 암호의 암호)에 대한 자리 표시자가 있습니다. 자리 표시자를 환경의 인증 요구 사항으로 바꿉니다.

Red Hat Virtualization Manager는 각 리소스의 id 속성에 대해 전역적으로 고유한 식별자(GUID)를 생성합니다. 이러한 예의 식별자 코드는 Red Hat Virtualization 환경의 식별자 코드와 다릅니다.

이 예제에는 기본 예외 및 오류 처리 논리만 포함되어 있습니다. SDK와 관련된 예외 처리에 대한 자세한 내용은 ovirtsdk.infrastructure.errors 모듈의 pydoc을 참조하십시오.

```
$ pydoc ovirtsdk.infrastructure.errors
```

3.2. 버전 4에서 RED HAT VIRTUALIZATION MANAGER 연결

Red Hat Virtualization Manager에 연결하려면 스크립트 시작 시 클래스를 가져와 ovirtsdk4.sdk 모듈에서 Connection 클래스 인스턴스를 생성해야 합니다.

```
import ovirtsdk4 as sdk
```

Connection 클래스의 생성자는 여러 인수를 사용합니다. 지원되는 인수는 다음과 같습니다.

url

Manager의 기본 URL(예: https://server.example.com/ovirt-engine/api)이 포함된 문자열 입니다.

사용자 이름

연결할 사용자 이름(예: admin@internal)을 지정합니다. 이 매개변수는 필수입니다.

암호

username 매개변수에서 제공하는 사용자 이름의 암호를 지정합니다. 이 매개변수는 필수입니다.

토큰

사용자 이름 및 암호 대신 API에 액세스하기 위한 선택적 토큰입니다. token 매개변수를 지정하지 않으면 SDK가 자동으로 생성됩니다.

insecure

서버의 TLS 인증서 및 호스트 이름을 확인할지 여부를 나타내는 부울 플래그입니다.

ca_file

신뢰할 수 있는 CA 인증서가 포함된 PEM 파일입니다. 서버에서 제공하는 인증서는 이러한 CA 인증서를 사용하여 확인합니다. ca_file 매개 변수가 설정되지 않은 경우 시스템 전체 CA 인증서 저장소가 사용됩니다.

debug

디버그 출력을 생성해야 하는지 여부를 나타내는 부울 플래그입니다. 값이 True 이고 log 매개변수가 None 이 아닌 경우 서버에서 전송된 데이터가 로그에 기록됩니다.

참고

사용자 이름과 암호는 디버그 로그에 기록되므로 주의해서 처리합니다.

압축은 디버그 모드에서 비활성화되어 있습니다. 즉, 디버그 메시지가 일반 텍스트로 전송됩니다.

log

로그 메시지가 기록될 로거입니다.

kerberos

기본 인증 대신 **Kerberos** 인증을 사용해야 하는지 여부를 나타내는 부울 플래그입니다.

timeout

응답을 대기하는 최대 시간(초)입니다. 값 **0** (기본값)은 영구적으로 기다리는 것을 의미합니다. 응답을 수신하기 전에 시간 초과가 만료되면 예외가 발생합니다.

compress

SDK가 서버에 압축된 응답을 보내도록 요청할지 여부를 나타내는 부울 플래그입니다. 기본값은 **True** 입니다. 이는 서버에 대한 힌트로, 이 매개 변수가 **True** 로 설정된 경우에도 압축되지 않은 데이터를 반환할 수 있습니다. 압축은 디버그 모드에서 비활성화되어 있습니다. 즉, 디버그 메시지가 일반 텍스트로 전송됩니다.

sso_url

서버의 기본 **SSO URL**이 포함된 문자열입니다. 기본 **SSO URL**은 **sso_url** 이 제공되지 않으면 **URL** 에서 계산됩니다.

sso_revoke_url

SSO 취소 서비스의 기본 **URL**이 포함된 문자열입니다. 이는 외부 인증 서비스를 사용하는 경우에만 지정해야 합니다. 기본적으로 이 **URL**은 **url** 매개변수 값에서 자동으로 계산되므로 **Manager**의 일부인 **SSO** 서비스를 사용하여 **SSO** 토큰 해지가 수행됩니다.

sso_token_name

SSO 서버에서 반환된 **JSON SSO** 응답의 토큰 이름입니다. 기본값은 **access_token** 입니다.

headers

모든 요청과 함께 전송되어야 하는 헤더가 있는 사전입니다.

연결

호스트에 열 최대 연결 수입니다. 값이 0 (기본값)이면 연결 수는 무제한입니다.

pipeline

응답을 기다리지 않고 HTTP 파이프라인에 넣을 최대 요청 수입니다. 값이 0 (기본값)이면 pipelining이 비활성화됩니다.

```
import ovirtsdk4 as sdk

# Create a connection to the server:
connection = sdk.Connection(
    url='https://engine.example.com/ovirt-engine/api',
    username='admin@internal',
    password='password',
    ca_file='ca.pem',
)

connection.test()

print("Connected successfully!")

connection.close()
```

지원되는 전체 방법 목록은 Manager 머신에서 ovirtsdk.api 모듈에 대한 문서를 생성할 수 있습니다.

```
$ pydoc ovirtsdk.api
```

3.3. 데이터 센터 나열

데이터 센터 컬렉션에는 환경의 모든 데이터 센터가 포함됩니다.

예 3.1. 데이터 센터 나열

이 예에서는 데이터 센터 컬렉션의 데이터 센터를 나열하고 컬렉션의 각 데이터 센터에 대한 몇 가지 기본 정보를 출력합니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

connection = sdk.Connection(
```

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

dcs_service = connection.system_service().dcs_service()

dcs = dcs_service.list()

for dc in dcs:
    print("%s (%s)" % (dc.name, dc.id))

connection.close()
```

Default 데이터 센터만 존재하고 활성화되지 않은 환경에서 예제에서는 텍스트를 출력합니다.

```
Default (00000000-0000-0000-0000-000000000000)
```

3.4. 클러스터 나열

클러스터 컬렉션에는 환경의 모든 클러스터가 포함되어 있습니다.

예 3.2. 클러스터 나열

이 예에서는 클러스터 컬렉션의 클러스터를 나열하고 컬렉션의 각 클러스터에 대한 몇 가지 기본 정보를 출력합니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

cls_service = connection.system_service().clusters_service()

cls = cls_service.list()

for cl in cls:
```

```
    print("%s (%s)" % (cl.name, cl.id))

connection.close()
```

Default 클러스터만 있는 환경에서 예제에서는 텍스트를 출력합니다.

```
Default (00000000-0000-0000-0000-000000000000)
```

3.5. 호스트 나열

hosts 컬렉션에는 환경의 모든 호스트가 포함됩니다.

예 3.3. 호스트 나열

이 예에서는 호스트 컬렉션 및 해당 ID에 있는 호스트를 나열합니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

host_service = connection.system_service().hosts_service()

hosts = host_service.list()

for host in hosts:
    print("%s (%s)" % (host.name, host.id))

connection.close()
```

하나의 호스트인 MyHost 만 연결된 환경에서는 예제에서는 텍스트를 출력합니다.

```
MyHost (00000000-0000-0000-0000-000000000000)
```

3.6. 논리적 네트워크 나열

networks 컬렉션에는 환경의 모든 논리적 네트워크가 포함됩니다.

예 3.4. 논리 네트워크 나열

이 예제에서는 네트워크 컬렉션의 논리적 네트워크를 나열하고 컬렉션의 각 네트워크에 대한 몇 가지 기본 정보를 출력합니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

nws_service = connection.system_service().networks_service()

nws = nws_service.list()

for nw in nws:
    print("%s (%s)" % (nw.name, nw.id))

connection.close()
```

기본 관리 네트워크만 있는 환경에서 예제에서는 텍스트를 출력합니다.

```
ovirtmgmt (00000000-0000-0000-0000-000000000000)
```

3.7. 가상 머신 및 총 디스크 크기 나열

vms 컬렉션에는 가상 머신에 연결된 각 디스크의 세부 정보를 설명하는 디스크 컬렉션이 포함되어 있습니다.

예 3.5. 가상 머신 및 총 디스크 크기 나열

이 예제에서는 가상 머신 목록과 총 디스크 크기를 바이트 단위로 출력합니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

vms_service = connection.system_service().vms_service()

virtual_machines = vms_service.list()

if len(virtual_machines) > 0:

    print("%-30s  %s" % ("Name", "Disk Size"))
    print("================================================")

    for virtual_machine in virtual_machines:
        vm_service = vms_service.vm_service(virtual_machine.id)
        disk_attachments = vm_service.disk_attachments_service().list()
        disk_size = 0
        for disk_attachment in disk_attachments:
            disk = connection.follow_link(disk_attachment.disk)
            disk_size += disk.provisioned_size

            print("%-30s: %d" % (virtual_machine.name, disk_size))
```

예에서는 가상 머신 이름 및 디스크 크기를 출력합니다.

```
Name                    Disk Size
================================================
vm1                     50000000000
```

3.8. NFS 데이터 스토리지 생성

Red Hat Virtualization 환경을 처음 생성하는 경우 데이터 스토리지 도메인과 ISO 스토리지 도메인을 정의해야 합니다. 데이터 스토리지 도메인은 가상 디스크를 저장하는 반면 ISO 스토리지 도메인은 게스트 운영 체제의 설치 미디어를 저장합니다.

storagedomains 컬렉션에는 환경의 모든 스토리지 도메인이 포함되어 있으며 스토리지 도메인을 추가하고 제거하는 데 사용할 수 있습니다.

참고

이 예에 제공된 코드에서는 원격 **NFS** 공유가 **Red Hat Virtualization**과 함께 사용하도록 사전 구성되어 있다고 가정합니다. **NFS** 공유 준비에 대한 자세한 내용은 *관리 가이드*를 참조하십시오.

예 3.6. NFS 데이터 스토리지 생성

이 예제에서는 **NFS** 데이터 도메인을 **storagedomains** 컬렉션에 추가합니다.

V4

V4의 경우 **add** 메서드는 새 스토리지 도메인을 추가하는 데 사용되며 **type** 클래스는 다음 매개 변수를 전달하는 데 사용됩니다.

- 스토리지 도메인의 이름입니다.
- 데이터 센터 개체는 데이터 센터 컬렉션에서 검색했습니다.
- 호스트 컬렉션에서 검색된 호스트 오브젝트입니다.
- 추가할 스토리지 도메인 유형(데이터,**iso** 또는 내보내기).
- 사용할 스토리지 형식입니다(**v1**,**v2** 또는 **v3**).

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

# Create the connection to the server:
connection = sdk.Connection(
    url='https://engine.example.com/ovirt-engine/api',
    username='admin@internal',
```

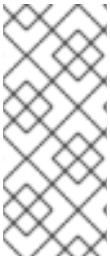
```
    password='password',
    ca_file='ca.pem',
)

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

# Create a new NFS storage domain:
sd = sds_service.add(
    types.StorageDomain(
        name='mydata',
        description='My data',
        type=types.StorageDomainType.DATA,
        host=types.Host(
            name='myhost',
        ),
        storage=types.HostStorage(
            type=types.StorageType.NFS,
            address='_FQDN_',
            path='/nfs/ovirt/path/to/mydata',
        ),
    ),
)

# Wait until the storage domain is unattached:
sd_service = sds_service.storage_domain_service(sd.id)
while True:
    time.sleep(5)
    sd = sd_service.get()
    if sd.status == types.StorageDomainStatus.UNATTACHED:
        break

print("Storage Domain '%s' added (%s)." % (sd.name(), sd.id()))

connection.close()
```

add 메서드 호출이 성공하면 예제에서는 텍스트를 출력합니다.

```
Storage Domain 'mydata' added (00000000-0000-0000-0000-000000000000).
```

3.9. NFS ISO 스토리지 생성

가상 머신을 생성하려면 게스트 운영 체제를 위한 설치 미디어가 필요합니다. 설치 미디어는 ISO 스토리지 도메인에 저장됩니다.



참고

이 예에 제공된 코드에서는 원격 **NFS** 공유가 **Red Hat Virtualization**과 함께 사용하도록 사전 구성되어 있다고 가정합니다. **NFS** 공유 준비에 대한 자세한 내용은 *관리 가이드*를 참조하십시오.

예 3.7. NFS ISO 스토리지 생성

이 예제에서는 **storagedomains** 컬렉션에 **NFS ISO** 도메인을 추가합니다.

V4

V4의 경우 **add** 메서드는 새 스토리지 도메인을 추가하는 데 사용되며 **type** 클래스는 다음 매개 변수를 전달하는 데 사용됩니다.

- 스토리지 도메인의 이름입니다.
- 데이터 센터 개체는 데이터 센터 컬렉션에서 검색했습니다.
- 호스트 컬렉션에서 검색된 호스트 오브젝트입니다.
- 추가할 스토리지 도메인 유형(데이터,**iso** 또는 내보내기).
- 사용할 스토리지 형식입니다(**v1**,**v2** 또는 **v3**).

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

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

```
# Use the "add" method to create a new NFS storage domain:
sd = sds_service.add(
    types.StorageDomain(
        name='myiso',
        description='My ISO',
        type=types.StorageDomainType.ISO,
        host=types.Host(
            name='myhost',
        ),
        storage=types.HostStorage(
            type=types.StorageType.NFS,
            address='FQDN',
            path='/nfs/ovirt/path/to/myiso',
        ),
    ),
)

# Wait until the storage domain is unattached:
sd_service = sds_service.storage_domain_service(sd.id)
while True:
    time.sleep(5)
    sd = sd_service.get()
    if sd.status == types.StorageDomainStatus.UNATTACHED:
        break

print("Storage Domain '%s' added (%s)." % (sd.name(), sd.id()))

# Close the connection to the server:
connection.close()
```

add 메서드 호출이 성공하면 예제에서는 텍스트를 출력합니다.

```
Storage Domain 'myiso' added (00000000-0000-0000-0000-000000000000).
```

3.10. 데이터 센터에 스토리지 도메인 연결

Red Hat Virtualization에 스토리지 도메인을 추가한 후 이를 데이터 센터에 연결하고 사용할 준비가 되기 전에 활성화해야 합니다.

예 3.8. 데이터 센터에 스토리지 도메인 연결

이 예에서는 기존 NFS 스토리지 도메인 mydata 를 기존 데이터 센터인 Default 에 연결합니다. 연결 작업은 데이터 센터의 storagedomains 컬렉션의 추가 방법으로 원활하게 수행됩니다. 이러한 예는 데이터와 ISO 스토리지 도메인을 모두 연결하는 데 사용할 수 있습니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

# Create the connection to the server:
connection = sdk.Connection(
    url='https://engine.example.com/ovirt-engine/api',
    username='admin@internal',
    password='password',
    ca_file='ca.pem',
)

# Locate the service that manages the storage domains and use it to
# search for the storage domain:
sds_service = connection.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 = connection.system_service().data_centers_service()
dc = dcs_service.list(search='name=Default')[0]

# Locate the service that manages the data center where we 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(
    types.StorageDomain(
        id=sd.id,
    ),
)

# Wait until the storage domain is active:
attached_sd_service = attached_sds_service.storage_domain_service(sd.id)
while True:
    time.sleep(5)
    sd = attached_sd_service.get()
    if sd.status == types.StorageDomainStatus.ACTIVE:
        break

print("Attached data storage domain '%s' to data center '%s' (Status: %s)." %
  (sd.name(), dc.name(), sd.status.state()))

# Close the connection to the server:
connection.close()
```

add 메서드 호출이 성공하면 예제에서는 다음 텍스트를 출력합니다.

```
Attached data storage domain 'data1' to data center 'Default' (Status: maintenance).
```

상태: 유지 관리는 스토리지 도메인을 계속 활성화해야 함을 나타냅니다.

3.11. 스토리지 도메인 활성화

Red Hat Virtualization에 스토리지 도메인을 추가하고 데이터 센터에 연결한 후에는 사용할 준비가 되기 전에 활성화해야 합니다.

예 3.9. 스토리지 도메인 활성화

이 예제에서는 데이터 센터인 Default 에 연결된 NFS 스토리지 도메인 mydata 를 활성화합니다. 활성화 작업은 스토리지 도메인의 활성화 방법으로 원활하게 수행됩니다.

V4

```
import ovirtsdk4 as sdk

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

# Locate the service that manages the storage domains and use it to
# search for the storage domain:
sds_service = connection.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 = connection.system_service().data_centers_service()
dc = dcs_service.list(search='name=Default')[0]

# Locate the service that manages the data center where we 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()
```

```
# Activate storage domain:
attached_sd_service = attached_sds_service.storage_domain_service(sd.id)
attached_sd_service.activate()

# Wait until the storage domain is active:
while True:
    time.sleep(5)
    sd = attached_sd_service.get()
    if sd.status == types.StorageDomainStatus.ACTIVE:
        break

print("Attached data storage domain '%s' to data center '%s' (Status: %s)." %
  (sd.name(), dc.name(), sd.status.state()))

# Close the connection to the server:
connection.close()
```

활성화 요청이 성공하면 예제에서는 텍스트가 출력됩니다.

```
Activated storage domain 'mydata' in data center 'Default' (Status: active).
```

status: active 는 스토리지 도메인이 활성화되었음을 나타냅니다.

3.12. ISO 스토리지 도메인의 파일 나열

storagedomains 컬렉션에는 스토리지 도메인의 파일을 설명하는 파일 컬렉션이 포함되어 있습니다.

예 3.10. ISO 스토리지 도메인의 파일 나열

이 예제에서는 각 ISO 스토리지 도메인의 ISO 파일 목록을 출력합니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

```
storage_domains_service = connection.system_service().storage_domains_service()

storage_domains = storage_domains_service.list()

for storage_domain in storage_domains:
   if(storage_domain.type == types.StorageDomainType.ISO):
      print(storage_domain.name + ":\n")
      files = storage_domain.files_service().list()

      for file in files:
         print("%s" % file.name + "\n")

connection.close()
```

다음은 텍스트를 출력하는 예입니다.

```
ISO_storage_domain:
file1
file2
```

3.13. 가상 머신 생성

가상 머신 생성은 여러 단계로 수행됩니다. 여기에서 다루는 첫 번째 단계는 가상 머신 오브젝트 자체를 생성하는 것입니다.

예 3.11. 가상 머신 생성

이 예제에서는 다음 요구 사항이 있는 가상 머신 **vm1** 을 생성합니다.

- 바이트로 표시된 **512MB**의 메모리입니다.
- **Default** 클러스터에 연결되므로 **Default** 데이터 센터에 연결합니다.
- 기본 **Blank** 템플릿 기반.
- 가상 하드 디스크 드라이브에서 부팅됩니다.

V4

V4에서는 add 메서드를 사용하여 옵션이 유형 로서 추가됩니다.

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

# 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(
    types.Vm(
        name='vm1',
        memory = 512*1024*1024
        cluster=types.Cluster(
            name='Default',
        ),
        template=types.Template(
            name='Blank',
        ),
        os=types.OperatingSystem(boot=types.Boot(devices=[types.BootDevice.HD)]
    ),
)

print("Virtual machine '%s' added." % vm.name)

# Close the connection to the server:
connection.close()
```

추가 요청이 성공하면 예제에 텍스트가 출력됩니다.

```
Virtual machine 'vm1' added.
```

3.14. 가상 NIC 생성

새로 생성된 가상 머신에 네트워크 액세스 권한이 있도록 하려면 가상 NIC를 생성하고 연결해야 합니다.

예 3.12. 가상 **NIC** 생성

이 예에서는 **NIC, NIC1** 을 생성하여 가상 머신 **vm1** 에 연결합니다. 이 예제의 **NIC**는 **virtio** 네트워크 장치이며 **NetNamespace** 관리 네트워크에 연결되어 있습니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

# 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=vm1')[0]

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

# Locate the vnic profiles service and use it to find the ovirmgmt
# network's profile id:
profiles_service = connection.system_service().vnic_profiles_service()
profile_id = None
for profile in profiles_service.list():
    if profile.name == 'ovirtmgmt':
        profile_id = profile.id
        break

# Use the "add" method of the network interface cards service to add the
# new network interface card:

nic = nics_service.add(
    types.Nic(
        name='nic1',
        interface=types.NicInterface.VIRTIO,
        vnic_profile=types.VnicProfile(id=profile_id),
    ),
)

print("Network interface '%s' added to '%s'." % (nic.name, vm.name))

connection.close()
```

추가 요청이 성공하면 예제에 텍스트가 출력됩니다.

```
Network interface 'nic1' added to 'vm1'.
```

3.15. 가상 머신 디스크 생성

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

예 3.13. 가상 머신 디스크 생성

이 예에서는 **8GB virtio** 디스크를 생성하여 가상 머신 **vm1** 에 연결합니다. 디스크에는 다음과 같은 요구 사항이 있습니다.

- **data1** 이라는 스토리지 도메인에 저장됩니다.
- **8GB** 크기.
- 시스템 유형 디스크(데이터와 반대).
- **virtio** 스토리지 장치.
- **COW** 형식.
- 사용 가능한 부팅 장치로 표시됩니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

connection = sdk.Connection(
    url='https://engine.example.com/ovirt-engine/api',
    username='admin@internal',
```

```
    password='password',
    ca_file='ca.pem',
)

# 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=vm1')[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(
    types.DiskAttachment(
        disk=types.Disk(
            format=types.DiskFormat.COW,
            provisioned_size=8*1024*1024,
            storage_domains=[
                types.StorageDomain(
                    name='data1',
                ),
            ],
        ),
        interface=types.DiskInterface.VIRTIO,
        bootable=True,
        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)
while True:
    time.sleep(5)
    disk = disk_service.get()
    if disk.status == types.DiskStatus.OK:
        break

print("Disk '%s' added to '%s'." % (disk.name(), vm.name()))

# Close the connection to the server:
connection.close()
```

추가 요청이 성공하면 예제에 텍스트가 출력됩니다.

```
Disk 'vm1_Disk1' added to 'vm1'.
```

3.16. 가상 머신에 ISO 이미지 연결

새로 생성된 가상 머신에 게스트 운영 체제를 설치하려면 운영 체제 설치 미디어가 포함된 ISO 파일을 연결해야 합니다. ISO 파일을 찾으려면 ISO 스토리지 도메인의 파일 목록을 참조하십시오.

예 3.14. 가상 머신에 ISO 이미지 연결

이 예제에서는 가상 시스템의 **cdroms** 컬렉션의 **add** 메서드를 사용하여 **my_iso_file.iso** 를 **vm1** 가상 시스템에 연결합니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

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

# Find the virtual machine:
vm = vms_service.list(search='name=vm1')[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 virtual machine:
cdroms_service = vm_service.cdroms_service()

# Get the first CDROM:
cdrom = cdroms_service.list()[0]

# Locate the service that manages the CDROM device found in previous step:
cdrom_service = cdroms_service.cdrom_service(cdrom.id)

# Change the CD of the VM to 'my_iso_file.iso'. By default the
# operation permanently changes the disk that is visible to the
# virtual machine after the next boot, but has no 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(
    cdrom=types.Cdrom(
        file=types.File(
            id='my_iso_file.iso'
```

```
        ),
    ),
    current=False,
)

print("Attached CD to '%s'." % vm.name())

# Close the connection to the server:
connection.close()
```

추가 요청이 성공하면 예제에 텍스트가 출력됩니다.

```
Attached CD to 'vm1'.
```

예 3.15. 가상 머신에서 cdrom 제거

이 예제에서는 가상 시스템의 cdrom 컬렉션에서 ISO 이미지를 제거합니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

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

# Find the virtual machine:
vm = vms_service.list(search='name=vm1')[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()

# Get the first found CDROM:
cdrom = cdroms_service.list()[0]

# Locate the service that manages the CDROM device found in previous step
# of the VM:
cdrom_service = cdroms_service.cdrom_service(cdrom.id)
```

```
cdrom_service.remove()

print("Removed CD from '%s'." % vm.name())

connection.close()
```

삭제 또는 제거 요청이 성공하면 예제에 텍스트가 출력됩니다.

```
Removed CD from 'vm1'.
```

3.17. 디스크 분리

가상 머신에서 디스크를 분리할 수 있습니다.

디스크 분리

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

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

# Find the virtual machine:
vm = vms_service.list(search='name=vm1')[0]

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

attachments_service = vm_service.disk_attachments_service()
attachment = next(
    (a for a in disk_attachments if a.disk.id == disk.id), None
)

# 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)
```

```
print("Detached disk %s successfully!" % attachment)

# Close the connection to the server:
connection.close()
```

삭제 또는 제거 요청이 성공하면 예제에 텍스트가 출력됩니다.

```
Detached disk vm1_disk1 successfully!
```

3.18. 가상 머신 시작

가상 머신을 시작할 수 있습니다.

예 3.16. 가상 머신 시작

이 예제에서는 시작 방법을 사용하여 가상 시스템을 시작합니다.

V4

```
import time
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

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

# Find the virtual machine:
vm = vms_service.list(search='name=vm1')[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 is up:
while True:
```

```
    time.sleep(5)
    vm = vm_service.get()
    if vm.status == types.VmStatus.UP:
        break

print("Started '%s'." % vm.name())

# Close the connection to the server:
connection.close()
```

시작 요청이 성공하면 예제에 텍스트가 출력됩니다.

```
Started 'vm1'.
```

UP 상태는 가상 머신이 실행 중임을 나타냅니다.

3.19. OVERRIDDEN 매개변수를 사용하여 가상 머신 시작

가상 머신을 시작하여 기본 매개변수를 재정의할 수 있습니다.

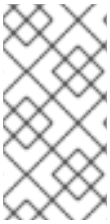
예 3.17. 재정의된 매개 변수를 사용하여 가상 머신 시작

이 예제에서는 Windows ISO를 사용하여 가상 머신을 부팅하고 Windows 드라이버가 포함된 virtio-win_x86.vfd 플로피 디스크를 연결합니다. 이 작업은 관리 포털에서 Run Once (한 번 실행) 창을 사용하여 가상 시스템을 시작하는 것과 동일합니다.

V4

```
import time
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

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

# Find the virtual machine:
```

```
vm = vms_service.list(search='name=vm1')[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 virtual machine:
cdroms_service = vm_service.cdroms_service()

# Get the first CDROM:
cdrom = cdroms_service.list()[0]

# Locate the service that manages the CDROM device found in previous step:
cdrom_service = cdroms_service.cdrom_service(cdrom.id)

# Change the CD of the VM to 'windows_example.iso':
cdrom_service.update(
   cdrom=types.Cdrom(
      file=types.File(
         id='windows_example.iso'
      ),
   ),
   current=False,
)

# Call the "start" method of the service to start it:
vm_service.start(
   vm=types.Vm(
      os=types.OperatingSystem(
         boot=types.Boot(
            devices=[
               types.BootDevice.CDROM,
            ]
         )
      ),
   )
)

# Wait until the virtual machine's status is "UP":
while True:
   time.sleep(5)
   vm = vm_service.get()
   if vm.status == types.VmStatus.UP:
      break

print("Started '%s'." % vm.name())

# Close the connection to the server:
connection.close()
```



참고

CD 이미지와 플로피 디스크 파일은 가상 머신에서 사용할 수 있어야 합니다. 자세한 내용은 데이터 스토리지 도메인에 이미지 업로드 를 참조하십시오.

3.20. CLOUD-INIT를 사용하여 가상 머신 시작

Cloud-Init 도구를 사용하여 특정 구성으로 가상 머신을 시작할 수 있습니다.

예 3.18. Cloud-Init를 사용하여 가상 머신 시작

이 예에서는 Cloud-Init 도구를 사용하여 가상 시스템을 시작하여 eth0 인터페이스에 호스트 이름과 고정 IP를 설정하는 방법을 보여줍니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

# Find the virtual machine:
vms_service = connection.system_service().vms_service()
vm = vms_service.list(search = 'name=vm1')[0]

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

# 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=types.Vm(
        initialization=types.Initialization(
        user_name='root',
        root_password='password',
        host_name='MyHost.example.com',
        nic_configurations=[
            types.NicConfiguration(
                name='eth0',
                on_boot=True,
                boot_protocol=types.BootProtocol.STATIC,
```

```
            ip=types.Ip(
                version=types.IpVersion.V4,
                address='10.10.10.1',
                netmask='255.255.255.0',
                gateway='10.10.10.1'
            )
        )
    )
)

# Close the connection to the server:
connection.close()
```

3.21. 시스템 이벤트 확인

Red Hat Virtualization Manager는 많은 시스템 이벤트를 기록하고 기록합니다. 이러한 이벤트 로그는 사용자 인터페이스, 시스템 로그 파일 및 API를 사용하여 액세스할 수 있습니다. ovirtsdk 라이브러리는 이벤트 컬렉션을 사용하여 이벤트를 노출합니다.

예 3.19. 시스템 이벤트 확인

이 예제에서는 이벤트 컬렉션이 나열됩니다.

list 메서드의 query 매개 변수는 사용 가능한 모든 결과 페이지가 반환되도록 하는 데 사용됩니다. 기본적으로 list 메서드는 100 레코드 길이의 결과 첫 번째 페이지만 반환합니다.

반환된 목록은 역방향 시간순으로 정렬되어 이벤트가 발생한 순서대로 표시됩니다.

V4

```
import ovirtsdk4 as sdk
import ovirtsdk4.types as types

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

# Find the service that manages the collection of events:
events_service = connection.system_service().events_service()
```

```
page_number = 1
events = events_service.list(search='page %s' % page_number)
while events:
   for event in events:
      print(
         "%s %s CODE %s - %s" % (
            event.time,
            event.severity,
            event.code,
            event.description,
         )
      )
   page_number = page_number + 1
   events = events_service.list(search='page %s' % page_number)

# Close the connection to the server:
connection.close()
```

다음 형식으로 출력 이벤트의 예는 다음과 같습니다.

```
YYYY-MM-DD_T_HH:MM:SS NORMAL CODE 30 - User admin@internal logged in.
YYYY-MM-DD_T_HH:MM:SS NORMAL CODE 153 - VM vm1 was started by admin@internal
(Host: MyHost).
YYYY-MM-DD_T_HH:MM:SS NORMAL CODE 30 - User admin@internal logged in.
```

부록 A. 법률 알림