Red Hat Virtualization 4.4

Python SDK ガイド

Red Hat Virtualization Python SDK の使用

Last Updated: 2023-08-22

Red Hat Virtualization 4.4 Python SDK ガイド

Red Hat Virtualization Python SDK の使用

Red Hat Virtualization Documentation Team
Red Hat Customer Content Services
rhev-docs@redhat.com

法律上の通知

概要

このガイドでは、Red Hat Virtualization Python ソフトウェア開発キットのバージョン 4 をインストールして操作する方法について説明します。

目次

第1章 概要

Python ソフトウェア開発キットのバージョン 4 は、Python ベースのプロジェクトで Red Hat Virtualization Manager と対話するためのクラスを集めたものです。これらのクラスをダウンロードしてプロジェクトに追加して、管理タスクの高レベルでの自動化を実現するため、さまざまな機能を利用できます。

注記

SDK バージョン 3 は、今後サポートされません。詳細については、このガイドの RHV4.3 バージョン を参照してください。

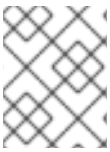
Python 3.7 と async

Python 3.7 以降のバージョンでは、**async** は予約済みのキーワードです。次の例のように、**async=True** が原因でエラーが発生するため、以前はサポート対象であったサービスのメソッドで、**async** パラメーターは使用できません。

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

これは、パラメーターにアンダースコアを追加する (**async_**) ことで解決できます。

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



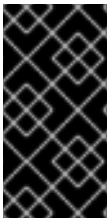
注記

この制限が該当するのは、Python 3.7 以降にのみです。前のバージョンの Python では、この変更は必要ありません。

1.1. 前提条件

Python ソフトウェア開発キットをインストールするには、以下が必要です。

- Red Hat Enterprise Linux 8 がインストールされているシステム。サーバーとワークステーションの両方のバリアントがサポートされています。
- Red Hat Virtualization エンタイトルメントのサブスクリプション。



重要

ソフトウェア開発キットは、Red Hat Virtualization REST API のインターフェイスです。お使いの Red Hat Virtualization 環境バージョンに対応するバージョンのソフトウェア開発キットを使用してください。たとえば、Red Hat Virtualization 4.3 を使用している場合は、V4 Python ソフトウェア開発キットを使用してください。

1.2. PYTHON ソフトウェア開発キットのインストール

Python ソフトウェア開発キットをインストールするには、以下を行います。

1. ハードウェアプラットフォームに適した リポジトリーを有効にします。たとえば、x86-64 ハードウェアの場合、次を有効にします。

```
# 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

# subscription-manager repos \
        --enable=rhel-8-for-x86_64-baseos-eus-rpms \
        --enable=rhel-8-for-x86_64-appstream-eus-rpms

# subscription-manager release --set=8.6
```

2. 必要なパッケージをインストールします。

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

Python ソフトウェア開発キットは Python 3 サイトパッケージディレクトリーにインストールされ、付属のドキュメントと例は **/usr/share/doc/python3-ovirt-engine-sdk4** にインストールされます。

第2章 ソフトウェア開発キットの使用

このセクションでは、バージョン 4 のソフトウェア開発キットの使用方法について説明します。

2.1. パッケージ

次のモジュールは、Python SDK で最も頻繁に使用されます。

ovirtsdk4

これはトップレベルのモジュールです。最も重要な要素は **Connection** クラスです。これは、サーバーに接続し、サービスツリーのルートへの参照を取得するためのメカニズムです。
Error クラスは、SDK がエラーを報告する必要があるときに発生するベース例外クラスです。

特定の種類のエラーには、ベースエラークラスを拡張する特定のエラークラスがあります。

- **AuthError** - 認証または認可が失敗したときに発生します。
- **ConnectionError** - サーバーの名前を解決できない場合、またはサーバーに到達できない場合に発生します。
- **NotFoundError** - リクエストされたオブジェクトが存在しない場合に発生します。
- **TimeoutError** - 操作がタイムアウトしたときに発生します。

ovirtsdk4.types

このモジュールには、API で使用されるタイプを実装するクラスが含まれています。たとえば、**ovirtsdk4.types.Vm** クラスは、仮想マシンタイプの実装です。これらのクラスはデータコンテナーであり、ロジックは含まれていません。
これらのクラスのインスタンスは、サービスメソッドのパラメーターおよび戻り値として使用されます。基礎となる表現への変換、基礎となる表現からの変換は、SDK によって透過的に処理されます。

ovirtsdk4.services

このモジュールには、API でサポートされているサービスを実装するクラスが含まれています。たとえば、**ovirtsdk4.services.VmsService** クラスは、システムの仮想マシンのコレクションを管理するサービスの実装です。
これらのクラスのインスタンスは、サービスが見つかると SDK によって自動的に作成されます。たとえば、次の手順を実行すると、**VmsService** クラスの新しいインスタンスが SDK によって自動的に作成されます。

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

コンストラクターのパラメーターをはじめ、一般的にはサービスロケーターやサービスメソッド以外のメソッドはすべて今後変更される可能性があるので、これらのクラスのインスタンスを手動で作成しないようにすることが最善です。

他にも、**ovirtsdk4.http**、**ovirtsdk4.readers**、**ovirtsdk4.writers** などのモジュールがあります。これらは、HTTP 通信の実装、および XML の解析とレンダリングに使用されます。これらは将来変更される可能性のある内部実装の詳細であるため、使用しないでください。下位互換性は保証されません。

2.2. サーバーへの接続

サーバーに接続するには、**Connection** クラスを含む **ovirtsdk4** モジュールをインポートします。これは SDK のエントリーポイントであり、API のサービスツリーのルートへのアクセスを提供します。

```
import ovirtsdk4 as sdk

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

この接続には、サーバーへの HTTP 接続のプールや認証トークンなどの重要なリソースが含まれています。これらのリソースが使用されなくなったら、これらのリソースを解放することが非常に重要です。

```
connection.close()
```

接続が閉じられると、再利用できなくなります。

TLS で保護されたサーバーに接続する場合は、**ca.pem** ファイルが必要です。通常のインストールでは、Manager マシンの **/etc/pki/ovirt-engine/** にあります。**ca_file** を指定しない場合、システム全体の CA 証明書ストアが使用されます。**ca.pem** ファイルの取得の詳細は、REST API ガイド を参照してください。

正常に接続されない場合には、SDK で、詳細を含む **ovirtsdk4.Error** の例外が発生します。

2.3. タイプの使用

ovirtsdk4.types モジュールのクラスは、純粋なデータコンテナーです。ロジックや操作は含まれません。タイプのインスタンスは、自由に作成および変更できます。

インスタンスの作成または変更は、以下で説明するサービスメソッドのいずれかを呼び出して変更が明示的に渡されない限り、サーバー側に影響を与えません。サーバー側での変更は、すでにメモリーに存在するインスタンスに自動的に反映されません。

これらのクラスのコンストラクターには、オプションの引数が複数 (タイプの属性ごとに 1 つずつ) あります。これは、複数のコンストラクターへのネストされた呼び出しを使用して、オブジェクトの作成を簡素化することを目的としています。この例では、仮想マシンのインスタンスを作成し、クラスター名、テンプレート、およびメモリーをバイト単位で指定します。

```
from ovirtsdk4 import types

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

このようにコンストラクターを使用することをお勧めしますが、必須ではありません。コンストラクターの呼び出しでインスタンスを引数を指定せずに作成し、セッターを使用してオブジェクトを段階的に設定することも、両方のアプローチを組み合わせて使用することも可能です。

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

API の仕様でオブジェクトのリストとして定義されている属性は、Python リストとして実装されます。たとえば、**Vm** タイプの **custom_properties** 属性は、**CustomProperty** タイプのオブジェクトのリストとして定義されます。SDK で属性が使用されている場合、それらは Python リストです。

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

API で列挙値として定義されている属性は、Python では **enum** として実装され、Python 3 では **enums** のネイティブサポート、Python 2.7 では **enum34** パッケージのネイティブサポートを使用して実装されます。この例では、**Vm** タイプのステータス属性は **VmStatus enum** を使用して定義されています。

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

注記

API 仕様では、XML と JSON に使用されるため、**enum** タイプの値は小文字で表示されます。しかし、Python の規則では **enum** 値は大文字になります。

タイプのインスタンスの属性の読み取りは、対応するプロパティーを使用して行われます。

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

2.4. リンクの使用

タイプの一部の属性は、API によってリンクとして定義されています。この規則は、そのオブジェクトの表現を取得するときに、値が通常は入力されないことを示しています。代わりにリンクが返されます。たとえば、仮想マシンを取得する場合、サーバーからの XML レスポンスには **<link>** 属性が含まれます。

```
<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 へのリンクには、実際のディスクアタッチメントは含まれません。データを取得するために、**Connection** クラスは、**href** XML 属性の値を使用して実際のデータを取得する **follow_link** メソッドを提供します。たとえば、仮想マシンのディスクの詳細を取得するには、ディスクアタッチメントへのリンクをたどり、次に各ディスクへのリンクをたどります。

```
# 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. サービスの場所

API は一連のサービスを提供し、それぞれがサーバーの URL スペース内のパスに関連付けられています。たとえば、システムの仮想マシンのコレクションを管理するサービスは **/vms** にあり、識別子 **123** の仮想マシンを管理するサービスは **/vms/123** にあります。

SDK では、そのサービスツリーのルートはシステムサービスによって実装されます。これは、接続の **system_service** メソッドを呼び出して取得されます。

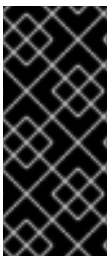
```
system_service = connection.system_service()
```

このシステムサービスへの参照がある場合、それを使用し、前のサービスのサービスロケーターと呼ばれる ***_service** メソッドを呼び出して、他のサービスへの参照を取得できます。たとえば、システムの仮想マシンのコレクションを管理するサービスへの参照を取得するには、**vms_service** サービスロケーターを使用します。

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

識別子 **123** の仮想マシンを管理するサービスへの参照を取得するには、仮想マシンのコレクションを管理するサービスの **vm_service** サービスロケーターを使用します。これは、仮想マシンの識別子をパラメーターとして使用します。

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



重要

サービスロケーターを呼び出しても、サーバーにリクエストは送信されません。返される Python オブジェクトは純粋なサービスであり、データは含まれません。たとえば、この例で呼び出される **vm_service** Python オブジェクトは、仮想マシンの表現ではありません。これは、仮想マシンの取得、更新、削除、開始、および停止に使用されるサービスです。

2.6. サービスの使用

サービスの場所を特定すると、そのサービスメソッドを呼び出すことができます。これにより、サーバーにリクエストが送信され、実際の作業が行われます。

単一のオブジェクトを管理するサービスは通常、**get**、**update**、および **remove** メソッドをサポートします。

オブジェクトのコレクションを管理するサービスは通常、**list** と **add** メソッドをサポートします。

どちらの種類のサービスも、特に単一のオブジェクトを管理するサービスは、追加のアクションメソッドをサポートできます。

2.6.1. get メソッドの使用

これらのサービスメソッドは、単一オブジェクトの表現を取得するために使用されます。次の例では、識別子が **123** の仮想マシンの表現を取得します。

```
# 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()
```

レスポンスは、対応するタイプのインスタンスです。この場合は、Python クラス **ovirtsdk4.types.Vm** のインスタンスです。

一部のサービスの **get** メソッドは、オブジェクトの表現を取得する方法や、複数ある場合にどの表現を取得するかを制御する追加のパラメーターをサポートします。たとえば、仮想マシンの現在の状態と、次に起動したときの状態は異なることもあるため、どちらかの状態を取得するとします。仮想マシンを管理するサービスの **get** メソッドは、**next_run** ブール値パラメーターをサポートします。

```
# 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)
```

詳細については、SDK の リファレンスドキュメント を参照してください。

何らかの理由でオブジェクトを取得できない場合、SDK は失敗の詳細とともに **ovirtsdk4.Error** 例外を発生させます。これには、オブジェクトが実際には存在しない状況も含まれます。**get** サービスメソッドを呼び出すと例外が発生するので注意してください。オブジェクトが存在しない場合でも、その呼び出しはサーバーにリクエストを送信しないため、サービスロケーターメソッドの呼び出しが失敗することはありません。以下に例を示します。

```
# 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. list メソッドの使用

これらのサービスメソッドは、コレクションのオブジェクトの表現を取得します。この例では、システムの仮想マシンの完全なコレクションを取得します。

```
# 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()
```

結果は、対応するタイプのインスタンスを含む Python リストになります。たとえば、この場合、結果はクラス **ovirtsdk4.types.Vm** のインスタンスのリストになります。

一部のサービスの **list** メソッドは、追加のパラメーターをサポートしています。たとえば、ほとんどすべてのトップレベルコレクションは、結果をフィルタリングするための **search** パラメーターや、サーバーから返される結果の数を制限するための **max** パラメーターをサポートしています。この例では、**my** で始まる仮想マシンの名前を取得し、結果数の上限は 10 です。

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

注記

すべての **list** メソッドがこれらのパラメーターをサポートしているわけではありません。一部の **list** メソッドは他のパラメーターをサポートします。詳細については、SDK の リファレンスドキュメント を参照してください。

何らかの理由で返された結果のリストが空の場合、戻り値は空のリストになります。**None** になることはありません。

結果の取得中にエラーが発生した場合、SDK は失敗の詳細を含む **ovirtsdk4.Error** 例外を発生させます。

2.6.3. add メソッドの使用

これらのサービスメソッドは、コレクションに新しい要素を追加します。追加するオブジェクトを記述する関連タイプのインスタンスを受け取り、それを追加するリクエストを送信し、追加されたオブジェクトを記述したタイプのインスタンスを返します。

この例では、**vm1** という新しい仮想マシンを追加しています。

```
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'
        )
    )
)
```

何らかの理由でオブジェクトを作成できない場合、SDK は失敗の詳細を含む **ovirtsdk4.Error** 例外を発生させます。**None** を返すことはありません。

重要

この **add** メソッドによって返される Python オブジェクトは、関連するタイプのインスタンスです。これはサービスではなく、データのコンテナーです。この特定の例では、返されるオブジェクトは **ovirtsdk4.types.Vm** クラスのインスタンスです。仮想マシンの作成後に、仮想マシンの取得や起動などの操作を実行する必要がある場合は、最初に仮想マシンを管理するサービスを見つけ、対応するサービスロケーターを呼び出す必要があります。

```
# 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()
```

オブジェクトは非同期で作成されます。新しい仮想マシンを作成する場合、**add** メソッドがレスポンスを返してから、仮想マシンが完全に作成されて、使用できるようになります。オブジェクトのステータスをポーリングして、オブジェクトが完全に作成されていることを確認することをお勧めします。仮想マシンの場合、ステータスが **DOWN** になるまで確認する必要があります。

```
# 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
```

get メソッドで、ループを使用してオブジェクトのステータスを取得すると、ステータス属性が確実に更新されます。

2.6.4. update メソッドの使用

これらのサービスメソッドは、既存のオブジェクトを更新します。実行する更新を記述した関連タイプのインスタンスを受け取り、それを更新するリクエストを送信し、更新されたオブジェクトを記述したタイプのインスタンスを返します。

この例では、仮想マシンの名前を **vm1** から **newvm** に更新します。

```
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'
    )
)
```

更新を実行する場合、オブジェクトの**完全な**表現を送信しないでください。更新する属性のみを送信します。以下は実行しないでください。

```
# 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)
```

完全な表現を送信すると、2 つの問題が発生します。

- サーバーが必要とするよりもはるかに多くの情報を送信しているため、リソースが無駄になります。
- サーバーは、変更するつもりがなかった属性も含めて、オブジェクトのすべての属性を更新しようとします。これにより、サーバー側でバグが発生する可能性があります。

一部のサービスの **update** メソッドは、更新方法や更新対象を制御する追加のパラメーターをサポートしています。たとえば、仮想マシンの現在の状態と、次に仮想マシンが起動したときに使用される状態のどちらかを更新するとします。仮想マシンを管理するサービスの **update** メソッドは、**next_run** ブール値パラメーターをサポートします。

```
# 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
)
```

何らかの理由で更新を実行できない場合、SDK は失敗の詳細を含む **ovirtsdk4.Error** 例外を発生させます。**None** を返すことはありません。

この update メソッドによって返される Python オブジェクトは、関連するタイプのインスタンスです。これはサービスではなく、データのコンテナーです。この特定の例では、返されるオブジェクトは **ovirtsdk4.types.Vm** クラスのインスタンスになります。

2.6.5. remove メソッドの使用

これらのサービスメソッドは、既存のオブジェクトを削除します。これらは単一のオブジェクトを管理するサービスのメソッドであるため、通常はパラメーターを取りません。したがって、サービスは削除するオブジェクトをすでに認識しています。

この例では、識別子が **123** の仮想マシンを削除します。

```
# 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()
```

一部のサービスの **remove** メソッドは、削除方法や削除対象を制御する追加のパラメーターをサポートしています。たとえば、**detach_only** ブール値パラメーターを使用して、ディスクを保持したまま仮想マシンを削除することができます。

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

オブジェクトが正常に削除された場合、**remove** メソッドは **None** を返します。削除されたオブジェクトは返されません。何らかの理由でオブジェクトを削除できない場合、SDK は失敗の詳細を含む **ovirtsdk4.Error** 例外を発生させます。

2.6.6. その他のアクションメソッドの使用

仮想マシンの停止や起動など、さまざまな操作を実行するサービスメソッドもあります。

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

これらのメソッドの多くには、操作を変更するパラメーターが含まれています。たとえば、仮想マシンを起動するメソッドは、**cloud-init** を使用して起動する場合、**use_cloud_init** パラメーターをサポートします。

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

ほとんどのアクションメソッドは、成功すると **None** を返し、失敗すると **ovirtsdk4.Error** を発生させます。いくつかの操作メソッドは値を返します。たとえば、ストレージドメインを管理するサービスには、そのストレージドメインがすでにデータセンターにアタッチされているかどうかを確認してブール値を返す、**is_attached** アクションメソッドがあります。

```
# 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():
    ...
```

SDK の リファレンスドキュメント で、各サービスでサポートされているアクションメソッド、それらが取るパラメーター、および返される値を確認してください。

2.7. 関連情報

詳細と例については、次のリソースを参照してください。

- V4 REST API ガイド
- Python SDK リファレンスドキュメント
- Python SDK の例

2.7.1. モジュールドキュメントの生成

次のモジュールのドキュメントを、pydoc を使用して生成できます。

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

ドキュメントは、**ovirt-engine-sdk-python** パッケージで提供されます。Manager マシンで次のコマンドを実行して、これらのドキュメントの最新バージョンを表示します。

```
$ pydoc [MODULE]
```

第3章 PYTHON の例

3.1. 概要

このセクションでは、Python SDK を使用して、基本的な Red Hat Virtualization 環境内に仮想マシンを作成する手順の例を説明します。

これらの例では、**ovirt-engine-sdk-python** パッケージによって提供される **ovirtsdkPython** ライブラリーを使用しています。このパッケージは、Red Hat Subscription Manager の **Red Hat Virtualization** サブスクリプションプールに接続されているシステムで利用できます。ソフトウェアをダウンロードするためのシステムのサブスクリプションに関する詳細は ソフトウェア開発キットのインストール を参照してください。

以下も必要になります。

- Red Hat Virtualization Manager のネットワークインストール。
- ネットワーク接続され、接続された Red Hat Virtualization ホスト。
- 仮想マシンにインストールするためのオペレーティングシステムを含む ISO イメージファイル。
- Red Hat Virtualization 環境を設定する論理オブジェクトと物理オブジェクトの両方に関する実用的な理解。
- Python プログラミング言語の実用的な理解。

例には、認証の詳細のプレースホルダー (ユーザー名は **admin@internal**、パスワードは **password**) が含まれます。プレースホルダーをご使用の環境の認証要件に置き換えてください。

Red Hat Virtualization Manager は、各リソースの **id** 属性に対してグローバルに一意の識別子 (GUID) を生成します。これらの例の識別子コードは、お使いの Red Hat Virtualization 環境の識別子コードとは異なります。

例には、基本的な例外およびエラー処理ロジックのみが含まれています。SDK に固有の例外処理の詳細については、**ovirtsdk.infrastructure.errors** モジュールの pydoc を参照してください。

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

3.2. バージョン 4 の RED HAT VIRTUALIZATION への接続

Red Hat Virtualization Manager に接続するには、スクリプトの開始時にクラスをインポートして、**ovirtsdk4.sdk** モジュールから **Connection** クラスのインスタンスを作成する必要があります。

```
import ovirtsdk4 as sdk
```

Connection クラスのコンストラクターは、いくつかの引数を取ります。サポートされている引数は次のとおりです。

url

https://server.example.com/ovirt-engine/api などの Manager のベース URL を含む文字列。

username

admin@internal など、接続するユーザー名を指定します。このパラメーターは必須です。

password

username パラメーターで指定されたユーザー名のパスワードを指定します。このパラメーターは必須です。

token

ユーザー名とパスワードの代わりに使用する、API にアクセスするためのオプションのトークン。**token** パラメーターが指定されていない場合、SDK は自動的に 1 つ作成します。

insecure

サーバーの TLS 証明書とホスト名をチェックする必要があるかどうかを示すブール値フラグ。

ca_file

信頼できる CA 証明書を含む PEM ファイル。サーバーによって提示された証明書は、これらの CA 証明書を使用して検証されます。**ca_file** パラメーターが設定されていない場合、システム全体の CA 証明書ストアが使用されます。

debug

デバッグ出力を生成する必要があるかどうかを示すブール値フラグ。値が **True** で、**log** パラメーターが **None** でない場合、サーバーとの間で送受信されるデータはログに書き込まれます。

注記

ユーザー名とパスワードはデバッグログに書き込まれるため、取り扱いには注意してください。

デバッグモードでは圧縮が無効になっています。つまり、デバッグメッセージはプレインテキストとして送信されます。

log

ログメッセージが書き込まれるロガー。

Kerberos

デフォルトの基本認証の代わりに Kerberos 認証を使用する必要があるかどうかを示すブール値フラグ。

timeout

レスポンスを待機する最大合計時間 (秒単位)。値が **0** (デフォルト) の場合、永久に待機することを意味します。レスポンスを受信する前にタイムアウトが期限切れになると、例外が発生します。

compress

SDK がサーバーに圧縮されたレスポンスを送信するようにリクエストするかどうかを示すブール値フラグ。デフォルトは **True** です。これはサーバーへのヒントであり、このパラメーターが **True** に設定されている場合でも非圧縮データを返す可能性があります。デバッグモードでは圧縮が無効になっています。つまり、デバッグメッセージはプレインテキストとして送信されます。

sso_url

サーバーのベース SSO URL を含む文字列。**sso_url** が指定されていない場合、デフォルトの SSO URL は **url** から計算されます。

sso_revoke_url

SSO 取り消しサービスのベース URL を含む文字列。これは、外部認証サービスを使用する場合にのみ指定する必要があります。デフォルトでは、この URL は **url** パラメーターの値から自動的に計算されるため、SSO トークンの取り消しは、Manager の一部である SSO サービスを使用して実行されます。

sso_token_name

SSO サーバーから返された JSONSSO レスポンスのトークン名。デフォルト値は **access_token** です。

ヘッダー

ヘッダー付きのディクショナリー。リクエストごとに送信する必要があります。

connections

ホストに対して開く接続の最大数。値が **0** (デフォルト) の場合、接続数は無制限です。

パイプライン

レスポンスを待たずに HTTP パイプラインに入れるリクエストの最大数。値が **0** (デフォルト) の場合、パイプライン処理は無効になります。

```
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()
```

サポートされているメソッドの完全なリストについては、Manager マシンで **ovirtsdk.api** モジュールのドキュメントを生成できます。

```
$ pydoc ovirtsdk.api
```

3.3. データセンターの一覧表示

datacenters コレクションには、環境内のすべてのデータセンターが含まれています。

例3.1 データセンターの一覧表示

この例では、**datacenters** コレクションにデータセンターを一覧表示し、コレクション内の各データセンターに関する一部の基本情報を出力します。

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 のデータセンターのみが存在し、それがアクティブ化されていない環境では、例は次のテキストを出力します。

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

3.4. クラスターの一覧表示

clusters コレクションには、環境内のすべてのクラスターが含まれます。

例3.2 クラスターの一覧表示

この例では、**clusters** コレクション内のクラスターを一覧表示し、コレクション内の各クラスターに関する一部の基本情報を出力します。

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 のクラスターのみが存在する環境では、例は次のテキストを出力します。

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

3.5. ホストのリスト

hosts コレクションには、環境内のすべてのホストが含まれます。

例3.3 ホストの一覧表示

この例では、**hosts** コレクション内のホストとその ID を一覧表示します。

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()
```

MyHost という 1 つのホストのみが接続されている環境では、例は次のテキストを出力します。

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

3.6. 論理ネットワークの一覧表示

networks コレクションには、環境内のすべての論理ネットワークが含まれます。

例3.4 論理ネットワークの一覧表示

この例では、**networks** コレクション内の論理ネットワークを一覧表示し、コレクション内の各ネットワークに関する一部の基本情報を出力します。

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()
```

デフォルトの管理ネットワークのみが存在する環境では、例は次のテキストを出力します。

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

3.7. 仮想マシンと合計ディスクサイズの一覧表示

vms コレクションには、仮想マシンに接続されている各ディスクの詳細を記述した **disks** コレクションが含まれています。

例3.5 仮想マシンと合計ディスクサイズの一覧表示

この例では、仮想マシンとその合計ディスクサイズ (バイト単位) を一覧表示します。

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))
```

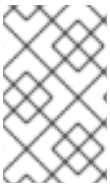
この例では、仮想マシン名とそのディスクサイズを出力します。

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

3.8. NFS データストレージの作成

Red Hat Virtualization 環境を最初に作成するときは、少なくともデータストレージドメインと ISO ストレージドメインを定義する必要があります。データストレージドメインは仮想ディスクを格納し、ISO ストレージドメインはゲストオペレーティングシステムのインストールメディアを格納します。

storagedomains コレクションには、環境内のすべてのストレージドメインが含まれており、ストレージドメインの追加と削除に使用できます。



注記

この例で提供されるコードは、リモート NFS 共有が Red Hat Virtualization で使用するために事前設定されていることを前提としています。NFS 共有の準備の詳細は、**管理ガイド** を参照してください。

例3.6 NFS データストレージの作成

この例では、NFS データドメインを **storagedomains** コレクションに追加します。

V4

V4 の場合、**add** メソッドを使用して新しいストレージドメインを追加し、**types** クラスを使用して次のパラメーターを渡します。

- ストレージドメインの名前。
- **datacenters** コレクションから取得したデータセンターオブジェクト。
- **host** コレクションから取得したホストオブジェクト。
- 追加されるストレージドメインのタイプ (**data**、**iso**、または **export**)。
- 使用するストレージ形式 (**v1**、**v2**、または **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 メソッドの呼び出しが成功すると、例は次のテキストを出力します。

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

3.9. NFS ISO ストレージの作成

仮想マシンを作成するには、ゲストオペレーティングシステム用のインストールメディアが必要です。インストールメディアは ISO ストレージドメインに保存されます。

注記

この例で提供されるコードは、リモート NFS 共有が Red Hat Virtualization で使用するために事前設定されていることを前提としています。NFS 共有の準備の詳細は、**管理ガイド** を参照してください。

例3.7 NFS ISO ストレージの作成

この例では、NFS ISO ドメインを **storagedomains** コレクションに追加します。

V4

V4 の場合、**add** メソッドを使用して新しいストレージドメインを追加し、**types** クラスを使用して次のパラメーターを渡します。

- ストレージドメインの名前。

- **datacenters** コレクションから取得したデータセンターオブジェクト。
- **host** コレクションから取得したホストオブジェクト。
- 追加されるストレージドメインのタイプ (**data**、**iso**、または **export**)。
- 使用するストレージ形式 (**v1**、**v2**、または **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 メソッドの呼び出しが成功すると、例は次のテキストを出力します。

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

3.10. ストレージドメインのデータセンターへのアタッチ

Red Hat Virtualization にストレージドメインを追加したら、そのドメインをデータセンターにアタッチし、アクティブ化してからでないと、使用できません。

例3.8 ストレージドメインのデータセンターへのアタッチ

この例では、既存の NFS ストレージドメインである **mydata** を、既存のデータセンターである **Default** に接続します。アタッチ操作は、データセンターの **storagedomains** コレクションの **add** メソッドによって容易になります。これらの例は、データと ISO ストレージドメインの両方をアタッチするために使用できます。

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 メソッドの呼び出しが成功すると、例は次のテキストを出力します。

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

Status: maintenance は、ストレージドメインをアクティブ化する必要があることを示します。

3.11. ストレージドメインのアクティブ化

ストレージドメインを Red Hat Virtualization に追加してデータセンターにアタッチしたら、使用できるようになる前にアクティブ化する必要があります。

例3.9 ストレージドメインのアクティブ化

この例では、データセンター (**Default**) にアタッチされている NFS ストレージドメイン (**mydata**) をアクティブにします。**activate** アクションは、ストレージドメインの **activate** メソッドによって容易になります。

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()
```

activate リクエストが成功すると、例は次のテキストを出力します。

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

Status: active は、ストレージドメインがアクティブ化されたことを示します。

3.12. ISO ストレージドメイン内のファイルの一覧表示

storagedomains コレクションには、ストレージドメイン内のファイルを記述する **files** コレクションが含まれています。

例3.10 ISO ストレージドメイン内のファイルの一覧表示

この例では、各 ISO ストレージドメインの ISO ファイルの一覧を出力します。

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()
```

この例では、テキストを出力します。

```
ISO_storage_domain:
file1
file2
```

3.13. 仮想マシンの作成

仮想マシンの作成は、いくつかの手順で実行されます。ここで説明する最初の手順は、仮想マシンオブジェクト自体を作成することです。

例3.11 仮想マシンの作成

この例では、次の要件を持つ仮想マシン **vm1** を作成します。

- 512MB のメモリー (バイト単位)。
- **Default** クラスターにアタッチされており、したがって **Default** データセンターにアタッチされている。
- デフォルトの **Blank** テンプレートに基づく。
- 仮想ハードディスクドライブから起動する。

V4

V4 では、オプションは **add** メソッドを使用して **types** として追加されます。

```
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()
```

add リクエストが成功すると、例は次のテキストを出力します。

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

3.14. 仮想 NIC の作成

新しく作成された仮想マシンがネットワークにアクセスできるようにするには、仮想 NIC を作成してアタッチする必要があります。

例3.12 仮想 NIC の作成

この例では、NIC (**nic1**) を作成し、それを仮想マシン (**vm1**) にアタッチします。この例の NIC は、**virtio** ネットワークデバイスであり、**ovirtmgmt** 管理ネットワークにアタッチされています。

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()
```

add リクエストが成功すると、例は次のテキストを出力します。

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

3.15. 仮想マシンディスクの作成

新しく作成された仮想マシンが永続ストレージにアクセスできるようにするには、ディスクを作成してアタッチする必要があります。

例3.13 仮想マシンディスクの作成

この例では、8 GB の **virtio** ディスクを作成し、仮想マシン **vm1** にアタッチします。ディスクには次の要件があります。

- **data1** という名前のストレージドメインに格納されている。
- サイズは 8 GB。
- **system** タイプのディスク (**data** ではない)。
- **virtio** ストレージデバイス。
- **COW** 形式。
- 使用可能なブートデバイスとしてマークされている。

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()
```

add リクエストが成功すると、例は次のテキストを出力します。

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

3.16. ISO イメージの仮想マシンへのアタッチ

新しく作成した仮想マシンにゲストオペレーティングシステムをインストールするには、オペレーティングシステムのインストールメディアを含む ISO ファイルをアタッチする必要があります。ISO ファイルを見つけるには、ISO ストレージドメイン内のファイルの一覧表示 参照してください。

例3.14 仮想マシンへの ISO イメージのアタッチ

この例では、仮想マシンの **cdroms** コレクションの **add** メソッドを使用して、**my_iso_file.iso** を **vm1** 仮想マシンにアタッチします。

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()
```

add リクエストが成功すると、例は次のテキストを出力します。

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

例3.15 仮想マシンからの CD-ROM の取り出し

この例では、仮想マシンの **cdrom** コレクションから ISO イメージを取り出します。

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()
```

delete または **remove** リクエストが成功すると、例は次のテキストを出力します。

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

3.17. ディスクのデタッチ

仮想マシンからディスクをデタッチできます。

ディスクのデタッチ

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()
```

delete または **remove** リクエストが成功すると、例は次のテキストを出力します。

```
Detached disk vm1_disk1 successfully!
```

3.18. 仮想マシンの起動

仮想マシンを起動できます。

例3.16 仮想マシンの起動

この例では、**start** メソッドを使用して仮想マシンを起動します。

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()
```

start リクエストが成功すると、例は次のテキストを出力します。

```
Started 'vm1'.
```

UP ステータスは、仮想マシンが実行中であることを示します。

3.19. パラメーターのオーバーライドによる仮想マシンの起動

デフォルトのパラメーターをオーバーライドして、仮想マシンを起動できます。

例3.17 パラメーターのオーバーライドによる仮想マシンの起動

この例では、Windows ISO を使用して仮想マシンを起動し、Windows ドライバーを含む **virtio-win_x86.vfd** フロッピーディスクをアタッチします。この操作は、管理ポータルの **Run Once** ウィンドウを使用して仮想マシンを起動するのと同じです。

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()
```

注記

CD イメージとフロッピーディスクファイルが仮想マシンで利用可能である必要があります。詳細は、データストレージドメインへのイメージのアップロード を参照してください。

3.20. CLOUD-INIT による仮想マシンの起動

Cloud-Init ツールを使用して、特定の設定で仮想マシンを起動できます。

例3.18 Cloud-Init による仮想マシンの起動

この例は、Cloud-Init ツールを使用し、**eth0** インターフェイスのホスト名と静的 IP を設定して仮想マシンを起動する方法を示しています。

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. システムイベントの確認

Red Hat Virtualization Manager は、多くのシステムイベントを記録およびログします。これらのイベントログには、ユーザーインターフェイス、システムログファイルからアクセスでき、API を使用してアクセスすることもできます。**ovirtsdk** ライブラリーは、**events** コレクションを使用してイベントを公開します。

例3.19 システムイベントの確認

この例では、**events** コレクションが一覧表示されます。

list メソッドの **query** パラメーターは、使用可能なすべての結果ページが返されるようにするために使用されます。デフォルトでは、**list** メソッドは結果の最初のページのみを返します。そのレコードの長さは **100** です。

返されたリストでは、新しく発生した順にイベントが表示されるよう並べ替えられます。

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()
```

これらの例は、次の形式でイベントを出力します。

```
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.
```

付録A 法的通知