Red Hat Virtualization 4.4

Python SDK 指南

使用 Red Hat Virtualization Python SDK

Last Updated: 2023-09-08

Red Hat Virtualization 4.4 Python SDK 指南

使用 Red Hat Virtualization Python SDK

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

法律通告

摘要

本指南论述了如何安装和使用 Red Hat Virtualization Python 软件开发工具包的版本 4。

目录

第 1 章 概述

Python 软件开发套件的版本 4 是一套类，可用于在基于 Python 的项目中与 Red Hat Virtualization Manager 交互。通过下载这些类并将它们添加到您的项目中，您可以访问一系列功能，以实现高级管理任务的自动化。

注意

SDK 的版本 3 不再被支持。如需更多信息，请参阅本指南 的 RHV 4.3 版本。

Python 3.7 和 async

在 Python 3.7 及更新的版本中，**sync** 是一个保留关键字。您不能在之前支持的服务方法中使用 **async** 参数，如下例所示，因为 **async=True** 会导致错误：

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

该解决方案是将下划线添加到参数(**async_**)：

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

注意

这个限制只适用于 Python 3.7 及更新的版本。早期版本的 Python 不需要进行此修改。

1.1. 先决条件

要安装 Python 软件开发工具包，您必须有：

- 安装 Red Hat Enterprise Linux 8 的系统。支持服务器和 Workstation 变体。
- Red Hat Virtualization 权利订阅。

重要

软件开发套件是 Red Hat Virtualization REST API 的界面。使用与您 Red Hat Virtualization 环境版本对应的软件开发组件版本。例如，如果您使用 Red Hat Virtualization 4.3，请使用 V4 Python 软件开发工具包。

1.2. 安装 PYTHON 软件开发套件

安装 Python 软件开发套件：

1. 启用 适合您的硬件平台 的仓库。例如，对于 x86-64 硬件，请启用：

```
# 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. 安装所需的软件包：

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

Python 软件开发套件安装到 Python 3 site-packages 目录中，并安装了附带的文档和示例到 **/usr/share/doc/python3-ovirt-engine-sdk4**。

第 2 章 使用软件开发套件

这部分论述了如何将软件开发套件用于版本 4。

2.1. 软件包

以下模块最常由 Python SDK 使用：

ovirtsdk4

这是顶级模块。最重要的元素是 **Connection** 类，这是连接到服务器的机制，并获取对服务树的根引用。
Error 类是 SDK 在需要报告错误时引发的基本例外类。

对于某些类型错误，有特定的错误类，它扩展了基本错误类：

- **AuthError** - 在身份验证或授权失败时 Raized。
- **ConnectionError** - 当服务器名称无法解析或者服务器无法访问时。
- **NotFoundError** - 当请求的对象不存在时 Raized。
- **TimeoutError** - 超时时 Raized。

ovirtsdk4.types

此模块包含实施 API 中使用的类型的类。例如，**ovirtsdk4.types.Vm** 类是虚拟机类型的实现。这些类是数据容器，不包含任何逻辑。
这些类的实例用作服务方法的参数和返回值。SDK 会透明地处理转换到底层表示的转换。

ovirtsdk4.services

此模块包含实施 API 支持的服务的类。例如，**ovirtsdk4.services.VmsService** 类是管理系统虚拟机集合的服务实现。
当服务位于时，SDK 会自动创建这些类的实例。例如，在执行以下操作时，SDD 会自动创建 **VmsService** 类的新实例：

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

最好避免手动创建这些类的实例，作为构造器的参数，一般情况下，除服务松cators 和服务方法以外的所有方法都可以在未来有所变化。

还有其他模块，如 **ovirtsdk4.http**、**ovirtsdk4.readers** 和 **ovirtsdk4.writers**。它们用于实施 HTTP 通信和 XML 解析和渲染。避免使用它们，因为它们是未来可能会变化的内部实施详情，因此无法保证向后兼容。

2.2. 连接到服务器

要连接到服务器，请导入包含 **Connection** 类的 **ovirtsdk4** 模块。这是 SDK 的入口点，并提供对 API 服务树的根访问：

```
import ovirtsdk4 as sdk

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

```
    username='admin@internal',
    password='password',
    ca_file='ca.pem',
)
```

连接保存关键资源，包括与服务器的 HTTP 连接和身份验证令牌等。在不再使用这些资源时，务必要释放这些资源：

```
connection.close()
```

连接关闭后，就无法重复使用。

当连接到使用 TLS 保护的服务器时，需要 **ca.pem** 文件。在正常安装中，它位于 Manager 机器的 **/etc/pki/ovirt-engine/** 中。如果没有指定 **ca_file**，则会使用系统范围的 CA 证书存储。有关获取 **ca.pem** 文件的更多信息，请参阅 *REST API 指南*。

如果连接不成功，则 SDK 会引发一个包含详情的 **ovirtsdk4.Error** 异常。

2.3. 使用类型

ovirtsdk4.types 模块中的类是纯数据容器。它们没有任何逻辑或操作。可以在 处创建和修改类型的实例。

创建或修改实例不会影响服务器端，除非通过以下其中一个服务方法明确传递了调用。服务器端的更改不会自动反映在内存中已存在的实例。

这些类的构造器具有多个可选参数，各自对应于 type 的各个属性。这是为了使用嵌套调用多个构造器来简化对象的创建。这个示例创建虚拟机实例，指定其集群名称、模板和内存，以字节为单位：

```
from ovirtsdk4 import types

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

建议以这种方式使用构造者，但不强制要求。您还可以在调用中不使用任何参数创建实例，并逐步填充对象步骤、使用 setter 或同时使用这两种方法的组合来填充对象：

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

在 API 规格中定义为对象列表的属性将作为 Python 列表实施。例如，**Vm** 类型的 **custom_properties** 属性定义为 **CustomProperty** 类型的对象列表。当在 SDK 中使用属性时，它们是一个 Python 列表：

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

在 Python 中以枚举值的形式定义的属性将作为 **enum** 实施，使用 Python 3 中的 **enum** 和 Python 2.7 中的 **enum34** 软件包的本地支持。在本例中，**Vm** 类型的 status 属性使用 **VmStatus enum** 来定义：

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

注意

在 API 规格中，enum 类型的值显示在小写中，因为这是用于 XML 和 JSON 什么。**但是，Python 惯例是大写 枚举 值。**

读取类型实例的属性是使用对应属性完成的：

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

2.4. 使用链接

API 作为链接定义的一些类型属性。此惯例表示在检索该对象的表示时通常不会填充这些值。相反，会返回一个链接。例如，在检索虚拟机时，来自服务器的 XML 响应包括 < link> 属性：

```
<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 的链接不包含实际的磁盘附加。要获取数据，Connection 类提供了一个 后续 _link 方法，它使用 href XML 属性的值来检索实际数据。例如，要检索虚拟机磁盘详情，请按照磁盘附加链接，然后访问每个磁盘：

```
# 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. 查找服务

API 提供了一组服务，每个服务都与服务器的 URL 空间中的路径相关联。例如，管理系统虚拟机集合的服务位于 /vms 中，而管理具有标识符 123 的虚拟机的服务则位于 /vms/123 中。

在 SDK 中，该服务树根由系统服务实施。它获取调用连接的 system_service 方法：

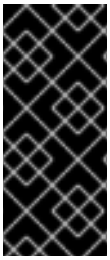
```
system_service = connection.system_service()
```

当您对此系统服务的引用时，您可以使用它来获取对其他服务的引用，调用之前服务的 *_service 方法（名为 service locators）。例如，若要获取管理系统虚拟机集合的服务的引用，您可以使用 vms_service 服务 locator：

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

要获取对使用标识符 123 管理虚拟机的服务的引用，您可以使用管理虚拟机集合的服务 vm_service 服务 locator。它使用虚拟机的标识符作为参数：

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



重要

调用服务 locators 不会向服务器发送请求。它们返回的 Python 对象是纯服务，不包含任何数据。例如，本例中调用的 vm_service Python 对象并不是虚拟机的表示。它是用于检索、更新、删除、启动和停止该虚拟机的服务。

2.6. 使用服务

找到服务后，您可以调用其服务方法，后者向服务器发送请求并执行实际工作。

管理单个对象的服务通常支持 获取、更新和删除 方法。

管理对象集合的服务通常支持 列表和 添加 方法。

这两种服务（特别是管理单个对象的服务）都可以支持其他操作方法。

2.6.1. 使用 get 方法

这些服务方法用于检索单个对象的表示。以下示例检索带有 123 的虚拟机的表示：

```
# 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()
```

响应是对应类型的实例，本例中为 Python 类 ovirtsdk4.types.Vm 的实例。

某些服务的 get 方法支持额外的参数，它们控制了如何检索对象的表示，或者检索对象的内容（如果存在多个参数）。例如，您可能要检索虚拟机的当前状态，或者下次启动时的状态，因为它们可能有所不同。管理虚拟机的服务的 get 方法支持 next_run 布尔值参数：

```
# 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)
```

详情请参阅 SDK 的 参考文档。

如果因为某种原因无法检索对象，则 SDK 会引发 ovirtsdk4.Error 异常，详细信息失败。这包括对象实际上不存在的情况。请注意，调用 get 服务方法时会引发异常。调用 service locator 方法永远不会失败，即使对象不存在，因为该调用不会向服务器发送请求。例如：

```
# 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. 使用 列表 方法

这些服务方法检索集合对象的表示。本例检索系统的虚拟机的完整集合：

```
# 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()
```

结果将是包含相应类型的实例的 Python 列表。例如，在这种情况下，结果将是类 ovirtsdk4.types.Vm 的实例列表。

一些 服务列表 方法支持额外的参数。例如，几乎所有顶级集合都支持 search 参数，用于过滤结果或 max 参数，以限制服务器返回的结果数。这个示例检索从 my 开始的虚拟机的名称，其上限为 10 个结果：

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

注意

并非所有 列表 方法都支持这些参数。些 列表 方法支持其他参数。详情请参阅 SDK 的参考文档。

如果因任何原因而返回的结果列表为空，则返回的值将是一个空列表。它绝不是 "无 "。

如果在尝试检索结果时出现错误，则 SDK 将引发一个 ovirtsdk4.Error 异常，其中包含失败的详细信息。

2.6.3. 使用 添加 方法

这些服务方法为集合添加新元素。它们收到描述要添加的对象的相关类型实例，发送请求来添加它，以及返回描述添加对象的类型实例。

此示例添加名为 vm1 的新虚拟机：

```
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'
        )
    )
)
```

如果因为某种原因无法创建对象，则 SDK 引发 ovirtsdk4.Error 异常，其中包含失败详情。它永远不会返回 "无 "。

重要

这个 add 方法返回的 Python 对象是相关类型的实例。它不是服务，而是数据容器。在这个特定示例中，返回的对象是 ovirtsdk4.types.Vm 类的实例。如果创建虚拟机后，您需要执行一个操作，如检索或启动它，您首先需要找到管理该服务，并调用对应的服务 locator：

```
# 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()
```

对象异步创建。在创建新虚拟机时，添加 方法将在虚拟机完全创建并准备好使用前返回响应。最好轮询对象的状态，以确保它完全创建。对于虚拟机，您应该检查其状态，直至状态为 DOWN ：

```
# 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
```

使用循环通过 get 方法检索对象状态，确保 status 属性已更新。

2.6.4. 使用 更新 方法

这些服务方法更新现有的对象。它们收到描述要执行更新的相关类型的实例，发送请求更新，再返回描述更新对象的类型实例。

这个示例将虚拟机的名称从 vm1 更新到 newvm ：

```
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'
    )
)
```

执行更新时，请避免发送对象 的完整 表示。仅发送您要更新的属性。不要：

```
# 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)
```

发送完整的表示会导致两个问题：

- 您发送的信息比服务器需求多得多，因此浪费资源。

- 服务器将尝试更新对象的所有属性，即使您没有更改对象的属性。这可能会导致服务器端的错误。

有些服务的 更新 方法支持额外的参数，它们控制如何或要更新的内容。例如，您可能想要更新虚拟机的当前状态，或者虚拟机下次启动时使用的状态。管理虚拟机的服务的更新 方法支持 next_run 布尔值参数：

```
# 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
)
```

如果因为某种原因无法更新，则 SDK 引发 ovirtsdk4.Error 异常，其中包含失败详情。它永远不会返回 "无 "。

此更新方法返回的 Python 对象是相关类型的实例。它不是服务，而是数据的一个容器。在这个特定示例中，返回的对象将是 ovirtsdk4.types.Vm 类的实例。

2.6.5. 使用 删除 方法

这些服务方法移除现有的对象。它们通常不使用参数，因为它们是管理单个对象的服务的方法。因此，该服务已经知道要删除的对象。

本例删除 ID 为 123 的虚拟机：

```
# 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()
```

删除 某些服务的方法支持额外的参数，它们控制如何删除。例如，可以在保留其磁盘的同时删除虚拟机，使用 detach_only 布尔值参数：

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

如果对象成功 移除，则删除方法会返回 None。它不会返回移除的对象。如果因为某种原因无法删除对象，则 SDK 引发 ovirtsdk4.Error 异常，其中包含失败详情。

2.6.6. 使用其他操作方法

还有其他服务方法执行各种操作，如停止和启动虚拟机：

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

其中许多方法包括修改操作的参数。例如，如果想要使用 cloud-init 启动虚拟机，则启动虚拟机的方法支持 use_cloud_init 参数：

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

大多数操作方法都会在成功时返回 None，并在失败时引发 ovirtsdk4.Error。少数操作方法返回值。例如，管理存储域的服务具有 is_attached 操作方法，用于检查存储域是否已附加到数据中心并返回布尔值：

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

检查 SDK 的 参考文档，以查看每个服务支持的操作方法、它们所使用的参数，以及它们返回的值。

2.7. 其它资源

有关详情和示例，请查看以下资源：

- V4 REST API 指南

- Python SDK 参考文档
- Python SDK 示例

2.7.1. 为模块生成文档

您可以使用以下模块使用 pydoc 生成文档：

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

文档由 ovirt-engine-sdk-python 软件包提供。在 Manager 计算机上运行以下命令，以查看这些文档的最新版本：

```
$ pydoc [MODULE]
```

第 3 章 PYTHON 示例

3.1. 概述

本节介绍了使用 Python SDK 在基本 Red Hat Virtualization 环境中创建虚拟机的步骤。

这些示例使用 ovirt-engine-sdk-python 软件包提供的 ovirtsdk Python 库。这个软件包适用于附加到 Red Hat Virtualization 订阅池的系统。有关 订阅您的系统以下载软件的更多信息，请参阅安装软件开发套件。

您还需要：

- Red Hat Virtualization Manager 的联网安装。
- 联网并配置的 Red Hat Virtualization 主机。
- 包含在虚拟机上安装的操作系统的 ISO 镜像文件。
- 需要了解组成 Red Hat Virtualization 环境的逻辑和物理对象。
- 需要了解 Python 编程语言。

示例包括用于身份验证详情的占位符（用户名admin@internal，密码为 password）。将占位符替换为您的环境的身份验证要求。

Red Hat Virtualization Manager 为每个资源的 id 属性生成全局唯一标识符(GUID)。这些示例中的标识符代码与 Red Hat Virtualization 环境中的标识符代码不同。

示例仅包含基本异常和错误处理逻辑。有关处理 SDK 异常的更多信息，请参阅 ovirtsdk.infrastructure.errors 模块的 pydoc：

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

3.2. 连接到版本 4 中的 RED HAT VIRTUALIZATION MANAGER

要连接到 Red Hat Virtualization Manager，您必须通过导入脚本开头的类从 ovirtsdk4.sdk 模块创建一个 Connection 类的实例：

```
import ovirtsdk4 as sdk
```

Connection 类的构造器采用几个参数。支持的参数有：

url

包含管理器基本 URL 的字符串，如 https://server.example.com/ovirt-engine/api。

username

指定用于连接的用户名，如 admin@internal。这个参数是必需的。

password

指定 username 参数提供的用户名的密码。这个参数是必需的。

token

用于访问 API 的可选令牌，而不是用户名和密码。如果没有指定 令牌 参数，则 SDK 将自动创建令牌参数。

insecure

指明是否应检查服务器的 TLS 证书和密钥名称的布尔值标志。

ca_file

包含可信 CA 证书的 PEM 文件。将使用这些 CA 证书来验证服务器所提供的证书。如果没有设置 ca_file 参数，则使用系统范围的 CA 证书存储。

debug

指明是否应该生成调试输出的布尔值标志。如果值为 True 且 log 参数不是 None，则从服务器发送和接收到的数据将写入到日志中。

注意

用户名和密码写入到调试日志中，因此要小心处理。

在调试模式中禁用压缩，这意味着调试消息会以纯文本形式发送。

log

将写入日志消息的日志记录器。

kerberos

指明是否应使用 Kerberos 验证的布尔值标志，而不是默认的基本身份验证。

timeout

响应等待的最长时间，以秒为单位。值 0（默认）表示要等待。如果超时在收到响应前过期，则会引发异常。

压缩

指明 SDK 是否应该要求服务器发送压缩响应的布尔值标志。默认值为 True。这是服务器的提示，它可以返回未压缩数据，即使此参数设为 True。在调试模式中禁用压缩，这意味着调试消息会以纯文本形式发送。

sso_url

包含服务器基本 SSO URL 的字符串。如果没有提供 sso_ url ，则默认 SSO URL 从 url 进行计算。

sso_revoke_url

包含 SSO 撤销服务的基本 URL 的字符串。这只有在使用外部身份验证服务时才需要指定。默认情况下，此 URL 会自动从 url 参数的值计算，以便将使用 SSO 服务（属于 Manager 的一部分）执行 SSO 令牌撤销。

sso_token_name

从 SSO 服务器返回的 JSON SSO 响应中的令牌名称。默认值为 access_token。

标头

包含标头的字典，应随每个请求一起发送。

连接

对主机打开的最大连接数。如果值为 0（默认），连接数量没有限制。

pipeline

要放入 HTTP 管道的最大请求数，而不等待响应。如果值为 0（默认），则 pipelining 被禁用。

```
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()
```

如需支持的方法的完整列表，您可以在 Manager 机器上生成 ovirtsdk.api 模块的文档：

```
$ pydoc ovirtsdk.api
```

3.3. 列出数据中心

数据中心 集合包含环境中的所有数据中心。

例 3.1. 列出数据中心

本例列出了数据中心收集中的 数据中心，并输出有关集合中每个数据中心的一些基本信息。

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()
```

在一个只存在 Default 数据中心的环境中，没有激活它，示例输出文本：

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

3.4. 列出集群

集群集合 包含环境中的所有集群。

例 3.2. 列出集群

本例列出了集群集合中的集群，并在集合中输出有关每个集群的一些基本信息。

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()
```

在只有 Default 集群存在的环境中，示例输出文本：

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

3.5. 列出主机

主机集合包含环境中的所有主机。

例 3.3. 列出主机

本例列出了主机集合及其 ID 中的主机。

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()
```

在仅附加了一台主机 MyHost 的环境中，示例输出文本：

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

3.6. 列出逻辑网络

网络 集合包含环境中的所有逻辑网络。

例 3.4. 列出逻辑网络

本例列出了网络集合中的逻辑网络，并在集合中输出有关每个网络的一些基本信息。

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()
```

在只存在默认管理网络的环境中，示例输出文本：

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

3.7. 列出虚拟机和总磁盘大小

vms 集合包含一个 磁盘 集合，用于描述附加到虚拟机的每个磁盘的详细信息。

例 3.5. 列出虚拟机以及总磁盘大小

这个示例打印虚拟机列表及其总磁盘大小（以字节为单位）：

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))
```

示例输出虚拟机名称及其磁盘大小：

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

3.8. 创建 NFS 数据存储

首次创建 Red Hat Virtualization 环境时，需要至少定义数据存储域和 ISO 存储域。数据存储域存储虚拟磁盘，而 ISO 存储域存储 guest 操作系统的安装介质。

storagedomains 集合包含环境中的所有存储域，并可用于添加和删除存储域。

注意

本例中提供的代码假定远程 NFS 共享已预先配置，可与 Red Hat Virtualization 一起使用。有关准备 NFS 共享的更多信息，请参阅 *管理指南*。

例 3.6. 创建 NFS 数据存储

这个示例在 storagedomains 集合中添加 NFS 数据域。

V4

对于 V4，可使用 add 方法添加新的存储域，并使用 类型 类传递以下参数：

- 存储域的名称。
- 从数据中心集合检索 的数据中心 对象。
- 从主机集合检索 的主机 对象。
- 正在添加的存储域的类型（数据、iso 或 导出）。
- 要使用的存储格式（v1、 v2 或 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:
```

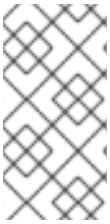
```
sd = sds_service.add(
   types.StorageDomain(
      name='mydata',
      description='My data',
      type=types.StorageDomainType.DATA,
      host=types.Host(
         name='myhost',
      ),
      storage=types.HostStorage(
         type=types.StorageType.NFS,
         address='_FQDN_',
         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()
```

如果 add 方法调用成功，则示例输出文本：

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

3.9. 创建 NFS ISO 存储

要创建虚拟机，您需要用于客户端操作系统的安装介质。安装介质保存在 ISO 存储域中。



注意

本例中提供的代码假定远程 NFS 共享已预先配置，可与 Red Hat Virtualization 一起使用。有关准备 NFS 共享的更多信息，请参阅 *管理指南*。

例 3.7. 创建 NFS ISO 存储

这个示例在 storagedomains 集合中添加 NFS ISO 域。

V4

对于 V4，可使用 add 方法添加新的存储域，并使用 类型 类传递以下参数：

- 存储域的名称。
- 从数据中心集合检索 的数据中心 对象。
- 从主机集合检索 的主机 对象。
- 正在添加的存储域的类型（数据、iso 或 导出）。
- 要使用的存储格式（v1、 v2 或 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()
```

如果 add 方法调用成功，则示例输出文本：

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

3.10. 将存储域附加到数据中心

将存储域添加到 Red Hat Virtualization 后，您必须将其附加到数据中心并在即将使用前将其激活。

例 3.8. 将存储域附加到数据中心

这个示例将现有的 NFS 存储域 mydata 附加到现有数据中心 Default。attach 操作可以通过数据中心的 storagedomains 集的 add 方法促进。这些示例可用于附加数据和 ISO 存储域。

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()
```

如果对 添加 方法的调用成功，示例输出以下文本：

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

Status：维护 表示仍需要激活存储域。

3.11. 激活存储域

将存储域添加到 Red Hat Virtualization 后，并将其附加到数据中心后，必须先将其激活后才能使用。

例 3.9. 激活存储域

本例激活了附加到数据中心 Default 的 NFS 存储域 mydata。激活 操作可以通过 存储域的激活 方法来实现。

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()
```

如果 激活 请求成功，示例输出文本：

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

Status：active 表示已激活存储域。

3.12. 列出 ISO 存储域中的文件

storagedomains 集合包含一个文件集合，用于描述存储域中的文件。

例 3.10. 列出 ISO 存储域中的文件

这个示例打印每个 ISO 存储域中的 ISO 文件列表：

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()
```

示例输出文本：

```
ISO_storage_domain:
file1
file2
```

3.13. 创建虚拟机

虚拟机创建在几个步骤中执行。此处涵盖的第一步是创建虚拟机对象本身。

例 3.11. 创建虚拟机

本例创建虚拟机 vm1，其要求如下：

- 512 MB 内存，以字节表示。
- 连接到 Default 集群，因此 Default 数据中心。
- 基于默认的 空白模板。
- 从虚拟硬盘引导。

V4

在 V4 中，使用 add 方法将选项添加为 类型。

```
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()
```

如果 添加 请求成功，则输出文本的示例：

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

3.14. 创建虚拟 NIC

要确保新创建的虚拟机具有网络访问权限，您必须创建并附加虚拟 NIC。

例 3.12. 创建虚拟 NIC

这个示例创建 NIC nic1，并将其附加到虚拟机 vm1。本例中的 NIC 是 virtio 网络设备，并附加到 ovirtmgmt 管理网络。

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 ovirtmgmt
# 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()
```

如果 添加 请求成功，则输出文本的示例：

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

3.15. 创建虚拟机磁盘

为确保新创建的虚拟机可以访问持久性存储，您必须创建并附加磁盘。

例 3.13. 创建虚拟机磁盘

这个示例创建了一个 8 GB VirtIO 磁盘，并将其附加到虚拟机 vm1。这个磁盘有以下要求：

- 存储在名为 data1 的存储域中。

- 大小为 8 GB。
- 系统 类型磁盘（而不是 数据）。
- VirtIO 存储设备.
- COW 格式。
- 标记为可用的引导设备。

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()
```

如果 添加 请求成功，则输出文本的示例：

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

3.16. 将 ISO 镜像附加到虚拟机

要在新创建的虚拟机上安装客户端操作系统，您必须附加包含操作系统安装介质的 ISO 文件。要找到 ISO 文件，请参阅 列出 ISO 存储域中的文件。

例 3.14. 将 ISO 镜像附加到虚拟机

本示例将 my_iso_file.iso 附加到 vm1 虚拟机，并使用虚拟机的 cdroms 集合的 add 方法。

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()
```

如果 添加 请求成功，则输出文本的示例：

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

例 3.15. 从虚拟机中弹出一个 CDROM

本例从虚拟机的 cdrom 集合中弹出 ISO 镜像。

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()
```

如果删除 或删除 请求成功，则示例输出文本：

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

3.17. 分离磁盘

您可以从虚拟机中分离磁盘。

分离磁盘

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()
```

如果删除 或删除 请求成功，则示例输出文本：

```
Detached disk vm1_disk1 successfully!
```

3.18. 启动虚拟机

您可以启动虚拟机。

例 3.16. 启动虚拟机

本示例使用 start 方法启动虚拟机。

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()
```

如果 启动 请求成功，则输出文本的示例：

```
Started 'vm1'.
```

UP 状态表示虚拟机正在运行。

3.19. 使用 OVERRIDDEN 参数启动虚拟机

您可以启动虚拟机覆盖其默认参数。

例 3.17. 使用覆盖的参数启动虚拟机

本示例使用 Windows ISO 引导虚拟机并附加 virtio-win_x86.vfd 软盘磁盘，其中包含 Windows 驱动程序。此操作等同于使用管理门户中的 Run Once 窗口启动虚拟机。

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()
```

注意

CD 映像和软盘文件必须可供虚拟机使用。详情请参阅将镜像上传到数据存储域。

3.20. 使用 CLOUD-INIT 启动虚拟机

您可以使用 Cloud-Init 工具启动使用特定配置的虚拟机。

例 3.18. 使用 Cloud-Init 启动虚拟机

本例演示了如何使用 Cloud-Init 工具启动虚拟机，以设置 eth0 接口的主机名和静态 IP。

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. 检查系统事件

Red Hat Virtualization Manager 记录并记录很多系统事件。这些事件日志可以通过用户界面、系统日志文件和 API 访问。ovirtsdk 库使用事件集合公开 事件。

例 3.19. 检查系统事件

本例中列出了 events 集合。

列表 方法的 query 参数用于确保返回所有可用的结果页面。默认情况下，列表 方法仅返回第一个

结果页面，其长度为 100 个记录。

返回的列表按相反顺序排序，按事件的发生顺序显示。

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()
```

这些示例以以下格式输出事件：

```
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.
```

附录 A. 法律通知