Red Hat Virtualization 4.4

Ruby SDK ガイド

Red Hat Virtualization Ruby SDK の使用

Last Updated: 2023-08-22

Red Hat Virtualization 4.4 Ruby SDK ガイド

Red Hat Virtualization Ruby SDK の使用

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

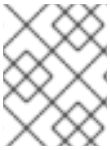
法律上の通知

概要

このガイドでは、Red Hat Virtualization Ruby ソフトウェア開発キットのインストール方法および操作方法について説明します。

目次

第1章 概要



注記

Ruby ソフトウェア開発キット (SDK) は非推奨になりました。Ruby SDK のサポートは、今後のリリースで削除される予定です。

Ruby ソフトウェア開発キットは、Ruby プロジェクトで Red Hat Virtualization Manager との対話を可能にする Ruby gem 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することにより、管理タスクを高レベルで自動化するためのさまざまな機能にアクセスできます。

1.1. 前提条件

Ruby 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。

1.2. RUBY ソフトウェア開発キットのインストール

1. 必要なリポジトリーを有効にします。

```
# 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
```

2. Ruby ソフトウェア開発キットをインストールします。

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

または、**gem** を使用してインストールすることもできます。

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

1.3. 依存関係

Ruby ソフトウェア開発キットには次の依存関係があります。**gem** を使用している場合は、これらを手動でインストールする必要があります。

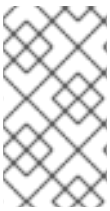
- XML の解析とレンダリング用の **libxml2**
- HTTP 転送用の **libcurl**
- C コンパイラー
- 必要なヘッダーファイルとライブラリーファイル

注記

RPM をインストールした場合は、依存関係ファイルをインストールする必要はありません。

依存関係ファイルをインストールします。

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



注記

Debian または Ubuntu を使用している場合は、**apt-get** を使用してください。

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

第2章 ソフトウェア開発キットの使用

この章では、Ruby ソフトウェア開発キットのモジュールとクラスを定義し、それらの使用法について説明します。

2.1. クラス

OvirtSDK4 モジュールには、次のソフトウェア開発キットクラスが含まれています。

接続

Connection クラスは、サーバーに接続し、サービスツリーの root への参照を取得するためのメカニズムです。詳細は、サーバーへの接続 を参照してください。

タイプ

Type クラスは、API でサポートされているタイプを実装します。たとえば、Vm クラスは仮想マシンタイプの実装です。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。タイプのインスタンスを操作します。
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、または基礎となる表現からの変換は、ソフトウェア開発キットによって透過的に処理されます。

サービス

Service クラスは、API でサポートされるサービスを実装します。たとえば、VmsService クラスは、システム内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、サービスが参照されるときに SDK によって自動的に作成されます。たとえば、**SystemService** クラスの **vms_service** メソッドを呼び出すと、SDK によって **VmsService** クラスの新しいインスタンスが自動的に作成されます。

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

警告

これらのクラスのインスタンスを手動で作成しないでください。コンストラクターのパラメーターとメソッドは、将来変更される可能性があります。

エラー

Error クラスは、ソフトウェア開発キットがエラーを報告すると発生するベース例外クラスです。特定のエラークラスは、ベースエラークラスを拡張します。

- AuthError - 認証または認可の失敗
- ConnectionError - サーバー名を解決できないか、サーバーに到達できません
- NotFoundError - 要求されたオブジェクトは存在しません
- TimeoutError - 操作のタイムアウト

他のクラス

他のクラス (たとえば、HTTP クライアントクラス、reader、writer) は、HTTP 通信および XML の解析とレンダリングに使用されます。これらのクラスには、将来変更される可能性のある内部実装の詳細が含まれるため、使用は推奨されません。それらの下位互換性は信頼できません。

2.2. タイプ

2.2.1. タイプのインスタンスの作成と変更

以下で説明するように、サービスメソッドを呼び出して変更がサーバーに明示的に送信されない限り、タイプのインスタンスを作成または変更しても、サーバー側には何の影響もありません。サーバー側での変更は、メモリーにすで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。これにより、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が簡素化されます。

次の例では、仮想マシンのインスタンスが作成され、その属性 (クラスター、テンプレート、およびメモリー) が設定されています。

属性を使用した仮想マシンインスタンスの作成

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

これらのコンストラクターに渡されたハッシュ (たとえば、**cluster: OvirtSDK4::Cluster.new**) は再帰的に処理されます。

次の例では、Cluster クラスと Template クラスのコンストラクターを明示的に呼び出す代わりに、プレーンなハッシュが使用されています。SDK は、ハッシュを必要なクラスに内部的に変換します。

プレーンなハッシュとして表現された属性を持つ仮想マシンインスタンスの作成

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

この方法で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。

次の例では、コンストラクターの呼び出しにおいて引数なしで仮想マシンインスタンスが作成されます。セッターを使用して、またはセッターとコンストラクターの組み合わせを使用して、仮想マシンインスタンスの属性を 1 つずつ追加できます。

仮想マシンインスタンスの作成と属性の個別の追加

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

API 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、Ruby 配列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。

属性のリストを配列として追加する

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

API で列挙値として定義されている属性は、列挙タイプと同じ名前のモジュールに定数として実装されます。

次の例は、**VmStatus** 列挙値を使用して **Vm** タイプのステータス属性を定義する方法を示しています。

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

重要

API 仕様では、**enum** タイプの値は小文字になります。これは、XML と JSON の規則であるためです。ただし、Ruby では、これらの定数には **大文字** を使用することが慣例となっています。

2.2.2. インスタンス属性の取得

対応する属性リーダーを使用して、インスタンス属性を取得できます。

次の例では、仮想マシンインスタンスの名前とメモリーを取得します。

仮想マシンインスタンス属性の取得

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

```
puts "vm.memory: #{vm.memory}"
vm.custom_properties.each do |custom_property|
  ...
end
```

リンクとしてのインスタンス属性の取得

一部のインスタンス属性はリンクとして返され、データを取得するには **follow_link** メソッドが必要です。次の例では、仮想マシンの属性の要求に対する応答は、リンク付きの XML としてフォーマットされています。

リンクとしての仮想マシン属性の取得

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

リンク **vm.disk_attachments** には、実際のディスクアタッチメントが含まれていません。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。

次の例で、**follow_link** を使用すると、ディスクの添付ファイル、そして各ディスクに移動して、**alias** を取得することができます。

仮想マシンサービスの取得

```
vm = vm_service.get
```

follow_link を使用したディスクアタッチメントとディスクエイリアスの取得

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

2.3. サービス

2.3.1. サービスの取得

API は、それぞれがサーバーパスに関連付けられた一連のサービスを提供します。たとえば、システム内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。

Ruby ソフトウェア開発キットでは、そのサービスツリーの root は **system service** によって実装されます。これは、接続の system_service メソッドを呼び出すことによって取得されます。

システムサービスの取得

```
system_service = connection.system_service
```

system service への参照を取得したら、それを使用して、***_service** メソッド (**service locators** と呼ばれる) により、他のサービスへの参照を取得できます。

たとえば、システム内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。

他のサービスの取得

```
vms_service = system_service.vms_service
```

識別子が **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、vm_service サービスのサービスロケーターを使用します。サービスロケーターは、仮想マシン識別子をパラメーターとして使用します。

識別子を使用した仮想マシンサービスの取得

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

重要

サービスロケーター呼び出しによって返されるオブジェクトは純粋なサービスであり、データは含まれていません。たとえば、前の例で取得した **vm_service** Ruby オブジェクトは、仮想マシンを表現するものではありません。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。

2.3.2. サービスメソッド

必要なサービスを見つけたら、そのサービスメソッドを呼び出すことができます。これらのメソッドはサーバーにリクエストを送信し、実際の作業を行います。

通常、オブジェクトのコレクションを管理するサービスには、**list** メソッドと **add** メソッドがあります。

単一のオブジェクトを管理するサービスには、通常、**get**、**update**、および **remove** メソッドがあります。

サービスには、取得、作成、更新、または削除以外のアクションを実行する追加のアクションメソッドがある場合があります。これらのメソッドは、単一のオブジェクトを管理するサービスで最も一般的に見られます。

2.3.2.1. Get

get メソッドは、単一のオブジェクトの表現を取得します。

次の例では、識別子が **123** の仮想マシンの表現を見つけて取得します。

```
# 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
```

結果は、対応するタイプのインスタンスになります。この場合、結果は Ruby クラス Vm のインスタンスになります。

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、複数の表現が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。

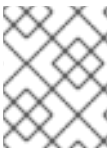
たとえば、起動後に仮想マシンの将来の状態を取得したい場合があります。仮想マシンを管理するサービスの **get** メソッドは、next_run ブール値パラメーターをサポートします。

仮想マシンの next_run 状態の取得

```
# 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、ソフトウェア開発キットの リファレンス ドキュメントを参照してください。

オブジェクトを取得できない場合、ソフトウェア開発キットはエラーの詳細を含む エラー 例外を発生させます。これは、存在しないオブジェクトを取得しようとした場合に発生します。



注記

service locator メソッドはサーバーにリクエストを送信しないため、オブジェクトが存在しない場合でも、**service locator** メソッドの呼び出しが失敗することはありません。

次の例では、**service locator** メソッドは成功しますが、**get** メソッドでは例外が発生します。

存在しない仮想マシンのサービスの検索: エラーなし

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

存在しない仮想マシンサービスの取得: エラー

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

2.3.2.2. List

list メソッドは、コレクション内の複数のオブジェクトの表現を取得します。

仮想マシンのコレクションの一覧表示

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

結果は、対応するタイプのインスタンスを含む Ruby 配列です。上記の例では、Ruby の Vm クラスのインスタンスの一覧が応答として返されます。

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** パラメーターと、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。

"my*" と呼ばれる 10 台の仮想マシンの一覧表示

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

注記

すべての list メソッドが **search** または **max** 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する場合があります。詳細は、リファレンス ドキュメントを参照してください。

結果のリストが空の場合、戻り値は空の Ruby 配列になります。**nil** になることはありません。

結果のリストを取得できない場合、SDK は失敗の詳細を含む エラー 例外を発生させます。

2.3.2.3. Add

add 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、それを追加する要求を送信し、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。

新しい仮想マシンの追加

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

重要

add メソッドによって返される Ruby 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、データの単なるコンテナーです。上記の例では、返されるオブジェクトは Vm クラスのインスタンスです。

追加した仮想マシンでアクションを実行する必要がある場合は、それを管理するサービスを見つけて、サービスロケーターを呼び出す必要があります。

新しい仮想マシンの起動

```
# 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
```

ほとんどのオブジェクトの作成は非同期タスクです。たとえば、新しい仮想マシンを作成する場合、**add** メソッドは、仮想マシンが完全に作成されて使用できるようになる前に、仮想マシンを返します。オブジェクトが完全に作成されるまで、オブジェクトのステータスをポーリングする必要があります。仮想マシンの場合、ステータスが **DOWN** になるまでチェックすることを意味します。

推奨されるアプローチは、仮想マシンを作成し、新しい仮想マシンを管理するサービスを見つけて、仮想マシンのステータスが **DOWN** になるまでステータスを繰り返し取得し、すべてのディスクが作成されたことを示すことです。

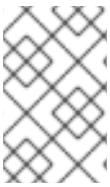
仮想マシンの追加、そのサービスの検索、およびそのステータスの取得

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

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

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

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 エラー 例外を発生させます。**nil** を返すことはありません。

2.3.2.4. Update

Update 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、それを更新する要求を送信し、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。



注記

この update メソッドによって返される Ruby 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、データの単なるコンテナーです。この特定の例では、返されるオブジェクトは Vm クラスのインスタンスになります。

次の例では、**service locator** メソッドが仮想マシンを管理しているサービスを検索し、**update** メソッドがその名前を更新します。

仮想マシン名の更新

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

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

オブジェクトを更新するときは、更新する属性のみを更新します。

仮想マシンの選択された属性の更新 (推奨)

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

オブジェクト全体を更新しないでください。

仮想マシンのすべての属性の更新 (非推奨)

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

仮想マシンのすべての属性を更新することはリソースの浪費であり、サーバー側で予期しないバグを引き起こす可能性があります。

一部のサービスの **Update** メソッドは、更新の方法または内容を制御するために使用できる追加のパラメーターをサポートします。たとえば、仮想マシンのメモリーを現在の状態ではなく、次に起動したときに更新したい場合があります。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。

次回実行時の仮想マシンのメモリー更新

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

更新を実行できない場合、SDK は失敗の詳細を含む エラー 例外を発生させます。**nil** を返すことはありません。

2.3.2.5. Remove

Remove 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り、サービスは削除するオブジェクトをすでに認識しているため、通常はパラメーターをサポートしません。

識別子が 123 の仮想マシンの削除

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

一部の **remove** メソッドは、削除する方法または内容を制御する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、ディスクを保持しながら仮想マシンを削除することができます。

ディスクを保持したままでの仮想マシンの削除

```
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 **nil** を返します。削除されたオブジェクトは返されません。

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 エラー 例外を発生させます。

2.3.2.6. 追加のアクション

上記の方法とは別に、追加のアクションメソッドがあります。仮想マシンを管理するサービスには、仮想マシンを開始および停止するメソッドがあります。

仮想マシンの起動

```
vm_service.start
```

一部のアクションメソッドには、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、**start** メソッドは use_cloud_init パラメーターをサポートします。

Cloud-Init による仮想マシンの起動

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

ほとんどのアクションメソッドは、成功すると **nil** を返し、失敗すると エラー を発生させます。ただし、一部のアクション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、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する **is_attached** アクションメソッドがあります。**is_attached** アクションメソッドはブール値を返します。

アタッチされているストレージドメインの確認

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

各サービスでサポートされているアクションメソッド、それらのパラメーター、および戻り値については、ソフトウェア開発キットの リファレンスドキュメント を参照してください。

第3章 RUBY の例

3.1. RED HAT VIRTUALIZATION MANAGER への接続

Connection クラスは、ソフトウェア開発キットのエントリーポイントです。Red Hat Virtualization Manager の REST API のサービスへのアクセスを提供します。

Connection クラスのパラメーターは次のとおりです。

- **url** - Red Hat Virtualization Manager API のベース URL
- **username**
- **password**
- **ca_file** - 信頼できる CA 証明書を含む PEM ファイル。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。**ca_file** を指定しない場合は、システム全体の CA 証明書ストアが使用されます。

Red Hat Virtualization Manager への接続

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

重要

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場合は解放する必要があります。

```
connection.close
```

接続、および接続から取得したすべてのサービスは、接続が閉じられた後は使用できません。

接続が失敗した場合、ソフトウェア開発キットは失敗の詳細を含む エラー 例外を発生させます。

詳細については、Connection:initialize を参照してください。

3.2. データセンターの一覧表示

この Ruby の例は、データセンターの一覧を表示します。

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

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

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

デフォルトのデータセンターしかない環境では、上記の例では以下が出力されます。

```
Default
```

詳細は、http://www.rubydoc.info/gems/ovirt-engine-sdk/OvirtSDK4/DataCentersService:list を参照してください。

3.3. クラスターの一覧表示

この Ruby の例は、クラスターの一覧を表示します。

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

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

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

デフォルトクラスターのみの環境では、上記の例では以下が出力されます。

```
Default
```

詳細については、ClustersService:list を参照してください。

3.4. 論理ネットワークの一覧表示

この Ruby の例は、論理ネットワークの一覧を表示します。

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

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

# Retrieve the list of clusters and for each one
# print its name:
nws = nws_service.list
```

```
nws.each do |nw|
  puts nw.name
end
```

デフォルトの管理ネットワークのみの環境では、上記の例では以下が出力されます。

```
ovirtmgmt
```

詳細については、NetworksService:list を参照してください。

3.5. ホストの一覧表示

この Ruby の例は、ホストの一覧を表示します。

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

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

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

ホストが 1 つだけアタッチされている環境 (**Atlantic**) では、上記の例では以下が出力されます。

```
Atlantic
```

詳細については、HostsService:list-instance_method を参照してください。

3.6. ISO ストレージドメインでの ISO ファイルの一覧表示

この Ruby の例では、ISO ストレージドメイン **myiso** 内の ISO ファイルの一覧が表示されます。

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

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

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

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

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

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

詳細については、FilesService:list を参照してください。

ISO ストレージドメインの名前がわからない場合、この Ruby の例ではすべての ISO ストレージドメインを取得します。

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

詳細については、StorageDomainsService:list を参照してください。

3.7. NFS ストレージドメインの作成

この Ruby の例では、NFS ストレージドメインを追加します。

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

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

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

# Wait until the storage domain is unattached:
sd_service = sds_service.storage_domain_service(sd.id)
loop do
  sleep(5)
  sd = sd_service.get
  break if sd.status == OvirtSDK4::StorageDomainStatus::UNATTACHED
end
```

詳細は、http://www.rubydoc.info/gems/ovirt-engine-sdk/OvirtSDK4/StorageDomainsService:add を参照してください。

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 Ruby の例では、既存の NFS ストレージドメイン **mydata** を既存のデータセンター **mydc** にアタッチします。この例は、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されます。

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

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

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

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

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

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

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

詳細は、http://www.rubydoc.info/gems/ovirt-engine-sdk/OvirtSDK4/StorageDomainsService:add を参照してください。

3.9. 仮想マシンの作成

この Ruby の例では、仮想マシンを作成します。この例では、値としてシンボルとネストされたハッシュを含むハッシュを使用しています。もう 1 つのメソッドはより詳細で、対応するオブジェクトのコンストラクターを直接使用します。詳細は、属性を使用した仮想マシンインスタンスの作成 を参照してください。

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

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

仮想マシンの作成後、仮想マシンのステータスをポーリング して、すべてのディスクが作成されたことを確認することをお勧めします。

詳細については、VmsService:add を参照してください。

3.10. VNIC プロファイルの作成

この Ruby の例では、vNIC プロファイルを作成します。

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

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

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

詳細については、VnicProfilesService:add を参照してください。

3.11. VNIC 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、vNIC を作成してアタッチする必要があります。

この Ruby の例では、vNIC を作成し、それを既存の仮想マシン **myvm** にアタッチします。

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

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

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

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

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

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

詳細については、VmsService:add を参照してください。

3.12. 仮想ディスクの作成

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。

この Ruby の例では、仮想ストレージディスクを作成して仮想マシンにアタッチします。

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

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

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

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

詳細については、DiskAttachmentsService:add を参照してください。

3.13. ISO イメージの仮想マシンへのアタッチ

この Ruby の例では、ゲストオペレーティングシステムをインストールするために、CD-ROM を仮想マシンにアタッチし、ISO イメージに変更します。

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

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

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

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

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

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

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

詳細については、VmService:cdroms_service を参照してください。

3.14. 仮想ディスクの取り外し

この Ruby の例では、仮想マシンからディスクをデタッチします。

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

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

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

詳細については、VmService:disk_attachments_service を参照してください。

3.15. 仮想マシンの起動

この Ruby の例では、仮想マシンを起動します。

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

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

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

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

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

詳細については、VmService:start を参照してください。

3.16. 起動デバイスと起動順序を指定した仮想マシンの起動

この Ruby の例では、起動デバイスと起動順序を指定して仮想マシンを起動します。

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

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

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

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

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

詳細については、BootDevice を参照してください。

3.17. CLOUD-INIT による仮想マシンの起動

この Ruby の例では、Cloud-Init ツールを使用して仮想マシンを起動し、**root** パスワードとネットワーク設定を設定します。

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

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

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

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

詳細については、VmService:start を参照してください。

3.18. システムイベントの確認

この Ruby の例では、ログに記録されたシステムイベントを取得します。

```
# In order to ensure that no events are lost, it is recommended to write
# the index of the last processed event, in persistent storage.
```

```
# Here, it is stored in a file called `index.txt`. In a production environment,
# it will likely be stored in a database.
INDEX_TXT = 'index.txt'.freeze

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

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

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

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

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

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

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

詳細については、EventsService:list を参照してください。

付録A 法的通知