pengxapi Documentation

Release 3.1.9

iOrchard

Jul 26, 2017

CONTENTS

Here is a pengx3 API pdf file. `pengx3.pdf`

Contents:

class api.pms.viewsets.**PMView**

index(*request*, *format=None*)
Get all PM info from DB.

- uri: /pms/all/
- method: GET
- request: None
- response:

```
<pms>{% for d in data %}
  <pm id="{{ d.uuid }}">
      <name>{{ d.name }}</name>
      <status>{{ d.status }}</status>
      <cpu_core>{{ d.cpu }}</cpu_core>
      <memory>{{ d.memory }}</memory>
      <interfaces>{% for network in d.network_set %}
          <interface>
              <name>{{ network.name }}</name>
              <mac>{{ network.mac|default_if_none:"" }}</mac>
              <ip>{{ network.ip.address|default_if_none:"" }}</ip>
          </interface>{% endfor %}
      </interfaces>
  </pm>{% endfor %}
</pms>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

model
alias of `PM`

read(*request*, *format=None*, *pm_uuid=None*)
Get PM info from DB.

- **uri:**

  /pms/ (Get myself info)
  /pms/:pm_uuid/ (Get pm_uuid info)

- method: GET
- request: None
- response:

```
<pm id="{{ data.uuid }}">
    <name>{{ data.name }}</name>
    <status>{{ data.status }}</status>
    <cpu_core>{{ data.cpu }}</cpu_core>
    <memory>{{ data.memory }}</memory>
    <interfaces>{% for network in data.network_set %}
        <interface>
```

```
            <name>{{ network.name }}</name>
            <mac>{{ network.mac|default_if_none:"" }}</mac>
            <ip>{{ network.ip.address|default_if_none:"" }}</ip>
        </interface>{% endfor %}
    </interfaces>
</pm>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

update_authorized_keys(*request*, *format=None*)

Update or create authorized_keys.

Import all ssh_pubkey of PM to authorized_keys

- uri: /pms/update-authorized-keys/
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

class api.pms.viewsets.**PerfView**

index(*request*, *format=None*)

Get the performance data of PM.

- uri: /pmperf/
- method: GET
- request: none
- response:

```
<pmperf id="{{ data.uuid }}">
  <status>Up</status>
  <vm_count>{{ data.vm_count }}</vm_count>
  <smt>{{ data.smt }}</smt>
  <total_cpu_cores>{{ data.cpus }}</total_cpu_cores>
  <used_cpu_cores>{{ data.used_cpu_cores }}</used_cpu_cores>
  <cpu_util>{{ data.cpu_util }} </cpu_util>
  <total_memory>{{ data.memory }}</total_memory>
  <used_memory>{{ data.used_memory }}</used_memory>
  <memory_util>{{ data.memory_util }}</memory_util>
  <iface_rx_sum>{{ data.iface_rx_sum }}</iface_rx_sum>
  <iface_tx_sum>{{ data.iface_tx_sum }}</iface_tx_sum>
</pmperf>
```

- **HTTP codes:**

success: HTTP_200_OK
failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

model
alias of PM

CHAPTER

ONE

STORLINK API

class api.storlinks.viewsets.**StorlinkView**

create(*request*, *format=None*)

Create a storlink to DB.

- uri: /storlinks/create/
- method: POST
- request:

```
<storlink>
  <name>store1</name>
  <link>/data1</link>
  <desc>VM store1</desc>
  <type>LO</type>
  <level>FL</level>
  <shared>True</shared>
</storlink>
```

- **field desc:**
  - link: storlink device path (/data, /dev/vgname)
  - **type: storlink type**

    LO Local disk

    FC Fiber Channel

    ISC ISCSI

    NFS Network File System

    CEP CEPH
  - **level: storlink base disk**

    FL File Level (NFS type should be FL)

    BL Block Level

    OB Object Level (CEPH type should be OB)
  - **shared: Shared storlink or not**

    True Shared storlink

    False non-shared storlink

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

delete(*request*, *format=None*, *id=None*)
Delete a storlink.

- uri: /storlinks/:id/
- method: DELETE
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_204_NO_CONTENT
  failure: HTTP_404_NOT_FOUND

model
alias of `Storlink`

read(*request*, *format=None*)
Read all Storlink instances from DB.

- uri: /storlinks/
- method: GET
- request: None
- response:

```
<storlinks>{% for storlink in data %}
  <storlink id="{{ storlink.id }}">
    <pm>{{ storlink.pm }}</pm>
    <name>{{ storlink.name }}</name>
    <link>{{ storlink.link }}</link>
    <type>{{ storlink.type }}</type>
    <level>{{ storlink.level }}</level>
    <size>{{ storlink.size }}</size>
    <free>{{ storlink.free }}</free>
    <desc>{{ storlink.desc }}</desc>
    <shared>{{ storlink.shared }}</shared>
  </storlink>{% endfor %}
</storlinks>
```

- **HTTP codes:**

success: HTTP_200_OK
failure: HTTP_404_NOT_FOUND

update(*request*, *format=None*, *id=None*)
Update storlink.

- uri: /storlinks/:id/
- method: PUT
- request:

```
<storlink>
  <name>store2</name>
  <link>/data2</link>
  <desc>VM store2</desc>
  <type>LO</type>
  <level>BL</level>
  <shared>False</shared>
</storlink>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

success: HTTP_204_NO_CONTENT
failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

class apiv2.storlinks.viewsets.**StorlinkViewV2**

read_all(*request*, *format=None*)
Read all Storlink instances from DB.

- api version : v2
- uri: /storlinks/all/
- method: GET
- request: None
- response:

```
<storlinks>{% for storlink in data %}
  <storlink id="{{ storlink.id }}">
    <pm>{{ storlink.pm }}</pm>
    <pm_name>{{ storlink.pm_name }}</pm_name>
    <name>{{ storlink.name }}</name>
    <link>{{ storlink.link }}</link>
    <type>{{ storlink.type }}</type>
    <level>{{ storlink.level }}</level>
    <size>{{ storlink.size }}</size>
    <free>{{ storlink.free }}</free>
    <desc>{{ storlink.desc }}</desc>
    <shared>{{ storlink.shared }}</shared>
```

```
  </storlink>{% endfor %}
</storlinks>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_404_NOT_FOUND

CHAPTER

TWO

TEMPLATE(GVM) API

class api.gvms.viewsets.**GVMView**

add(*request*, *format=None*)

Add a pre-built template into the DB.

- uri: /templates/add/
- method: POST
- request:

```
<template>
  <name>debian 8 64bit template</name>
  <os>DEBIAN</os>
  <os_arch>X64</os_arch>
  <desc>Debian 8 jessie 64bit template</desc>
  <disk>
    <storlink>11</storlink>
    <uuid>bed05d84-7c44-4720-9588-57cbd8ecef8c</uuid>
    <format>qcow2</format>
  </disk>
</template>
```

- **field desc:**
  - **os: os name**

    ARCH Arch Linux

    CENTOS CentOS

    COREOS CoreOS

    DEBIAN Debian GNU/Linux

    FEDORA Fedora

    GENTOO Gentoo Linux

    RHEL Redhat Enterprise Linux

    UBUNTU Ubuntu

    WINDOWS Windows

    OTHER Other
  - **os_arch: os architecture**

    X86 32bit x86

X64 64bit x86_64

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

create(*request*, *format=None*, *vm_uuid=None*)

Create a GVM instance from the VM instance.

- uri: /templates/create/vms/:vm_uuid/
- method: POST
- request:

```
<template>
  <name>$template_name</name>
  <storlink>2</storlink>
  <desc>$desc</desc>
</template>
```

- **process:**

  1. request data is valid checked
  2. create template disk
  3. create template db instance

- response:

```
<template id="{{data.uuid}}">
  <name>{{ data.name }}</name>
  <size>{{ data.size }}</size>
  <desc>{{ data.desc|default_if_none:"" }}</desc>
</template>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_400_BAD_REQUEST

delete(*request*, *format=None*, *gvm_uuid=None*)

Delete a GVM instance.

- uri: /template/:gvm_uuid/
- method: DELETE
- request: None
- **process:**

  1. **if template is linked to VM db:**

     – you could not delete the template.

2. **else:**
   - delete the template disks
   - delete the template db instance.

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_204_NO_CONTENT
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

disk_index(*request*, *format=None*, *gvm_uuid=None*)

Get template disks.

- uri: /templates/:gvm_uuid/disks/
- method: GET
- request: None
- response:

```
<disks>{% for d in data %}
  <disk id="{{ d.uuid }}">
    <name>{{ d.uuid }}</name>
    <order>{{ d.order }}</order>
    <size>{{ d.size }}</size>
    <template id="{{ d.vm }}"></template>
    <desc>{{ d.desc|default_if_none:"" }}</desc>
    <created>{{ d.created_at }}</created>
    <format>{{ d.format }}</format>
    <type>{{ d.type|default_if_none:"" }}</type>
    <storlink>{{ d.storlink }}</storlink>
  </disk>{% endfor %}
</disks>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

index(*request*, *format=None*)

Retrieve all GVM instances from DB.

- uri: /templates/
- method: GET
- request: None
- response:

```
<templates>{% for gvm in data %}
  <template id="{{gvm.uuid}}">
    <name>{{ gvm.name }}</name>
    <size>{{ gvm.size }}</size>
```

```
    <os>{{ gvm.os }}</os>
    <os_arch>{{ gvm.os_arch }}</os_arch>
    <desc>{{ gvm.desc|default_if_none:"" }}</desc>
  </template>{% endfor %}
</templates>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_400_BAD_REQUEST

model
alias of GVM

update(*request*, *format=None*, *gvm_uuid=None*)
Update a pre-built template info in the DB.

- uri: /templates/:gvm_uuid/
- method: PUT
- request:

```
<template>
  <name>debian 8 64bit template</name>
  <os>DEBIAN</os>
  <os_arch>X64</os_arch>
  <desc>Debian 8 jessie 64bit template</desc>
</template>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_204_NO_CONTENT
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

class apiv2.gvms.viewsets.**GVMViewV2**

add_with_new_image(*request*, *format=None*)
Create a template image and Add a template into the DB.

- uri: /templates/add_new/
- method: POST
- request:

```
<template>
  <name>debian 8 64bit template</name>
  <os>DEBIAN</os>
  <os_arch>X64</os_arch>
  <desc>Debian 8 jessie 64bit template</desc>
  <disk>
    <storlink>11</storlink>
```

```
    <size>102400<size> <!-- unit: MB -->
    <format>qcow2</format>
  </disk>
</template>
```

- **field desc:**
  - **os: os name**

    ARCH Arch Linux

    CENTOS CentOS

    COREOS CoreOS

    DEBIAN Debian GNU/Linux

    FEDORA Fedora

    GENTOO Gentoo Linux

    RHEL Redhat Enterprise Linux

    UBUNTU Ubuntu

    WINDOWS Windows

    OTHER Other
  - **os_arch: os architecture**

    X86 32bit x86

    X64 64bit x86_64
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

index(*request*, *format=None*)

Retrieve all GVM instances from DB.

- uri: /templates/
- method: GET
- request: None
- response:

```
<templates>{% for gvm in data %}
  <template id="{{gvm.uuid}}">
    <name>{{ gvm.name }}</name>
    <size>{{ gvm.size }}</size>
    <os>{{ gvm.os }}</os>
    <os_arch>{{ gvm.os_arch }}</os_arch>
    <desc>{{ gvm.desc|default_if_none:"" }}</desc>
```

```
    <lock_status>{{ gvm.lock_statue }}</lock_status>
    <pm>{{ gvm.pm}}</pm>
    <state>{{ gvm.state }}</state>
    <recent_booted_at>{{ gvm.recent_booted_at }}</recent_booted_at>
  </template>{% endfor %}
</templates>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_400_BAD_REQUEST

retrieve(*request*, *format=None*, *gvm_uuid=None*)

Retrieve all GVM instances from DB.

- uri: /templates/:gvm_uuid/
- method: GET
- request: None
- response:

```
<template id="{{gvm.uuid}}">
  <name>{{ gvm.name }}</name>
  <size>{{ gvm.size }}</size>
  <os>{{ gvm.os }}</os>
  <os_arch>{{ gvm.os_arch }}</os_arch>
  <desc>{{ gvm.desc|default_if_none:"" }}</desc>
  <lock_status>{{ gvm.lock_statue }}</lock_status>
  <pm>{{ gvm.pm}}</pm>
  <state>{{ gvm.state }}</state>
  <recent_booted_at>{{ gvm.recent_booted_at }}</recent_booted_at>
  <vnc_port>{{ gvm.vnc_port }}</vnc_port>
  <vnc_password>{{ gvm.vnc_port }}</vnc_password>
</template>{% endfor %}
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_400_BAD_REQUEST

CHAPTER

THREE

VM API

class api.vms.viewsets.**VMView**

copy(*request*, *format=None*, *vm_uuid=None*)

Copy VM.

- uri: /vms/:vm_uuid/copy/
- method: POST
- **process:**

  - If VM is running, then go to online copy else offline copy.
  - online copy
    - snapshot VM using qmp (Run Snapshot only)
    - VM is now read-only.
    - Copy VM using qemu-img convert.
    - Commit Run snapshot into VM.
  - offline copy
    - Copy VM using qemu-img convert.

- request:

```
<vm>
  <name>copied_vm</name>
  <storlink>2</storlink>
</vm>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_400_BAD_REQUEST

create(*request*, *format=None*)

Create a VM instance from the template or with empty disk.

- uri: /vms/
- method: POST

- request:

```
<vm>
  <name>test_vm</name>
  <template>$template_uuid</template>
  <storlink>$storlink_id</storlink>
  <cpu_core>1</cpu_core>
  <memory>512</memory>
  <maxmem>512</maxmem>
  <os_disk_size></os_disk_size>
  <extra_disk_size></extra_disk_size>
  <disk_format>qcow2</disk_format>
  <network>
    <list-item>0</list-item>
    <list-item>1</list-item>
  </network>
  <desc>VM for something</desc>
</vm>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

delete(*request*, *format=None*, *vm_uuid=None*)

Delete a VM instance.

- uri: /vms/:vm_uuid/
- method: DELETE
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_204_NO_CONTENT
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

index(*request*, *format=None*)

List all VM instances.

- uri: /vms/
- method: GET
- request: None
- response:

```
<vms>
  <vm id='{{ vm.uuid }}'>
    <name>{{ vm.name }}</name>
    <status>{{ vm.status }}</status>
    <cpu_core>{{ vm.cpu }}</cpu_core>
    <memory>{{ vm.memory }}</memory>
    <maxmem>{{ vm.maxmem }}</maxmem>
    <disk_size>{{ vm.disk_size }}</disk_size><!-- unit: MB -->
    <pm id='{{ vm.pm }}'></pm>
    <nics>
      <nic>
        <name>{{ net.name }}</name>
        <mac>{{ net.mac|default_if_none:"" }}</mac>
        <network>{{ net.switch_name }}</network>
      </nic>{% endfor %}
    </nics>
    <os>{{ vm.os|default_if_none:"" }}</os>
    <os_arch>{{ vm.os_arch|default_if_none:"" }}</os_type>
    <create_date>{{ vm.created_at }}</create_date>
    <last_boot_time>{{ vm.recent_booted_at }}</last_boot_time>
  </vm>
</vms>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

model
alias of VM

retrieve(*request*, *format=None*, *vm_uuid=None*)
Retrieve VM info.

- uri: /vms/:vm_uuid/
- method: GET
- request: None
- response:

```
<vm id='{{ data.uuid }}'>
  <name>{{ data.name }}</name>
  <status>{{ data.status }}</status>
  <cpu_core>{{ data.cpu }}</cpu_core>
  <memory>{{ data.memory }}</memory>
  <maxmem>{{ data.maxmem }}</maxmem>
  <disk_size>{{ data.disk_size }}</disk_size><!-- unit: MB -->
  <pm id='{{ data.pm }}'></pm>
  <nics>{% for net in data.vmiface_set %}
    <nic>
      <name>{{ net.name }}</name>
      <mac>{{ net.mac|default_if_none:"" }}</mac>
      <network>{{ net.switch_name }}</network>
    </nic>{% endfor %}
  </nics>
  <os>{{ data.os|default_if_none:"" }}</os>
  <os_type>{{ data.os_type|default_if_none:"" }}</os_type>
  <create_date>{{ data.created_at }}</create_date>
```

```
  <last_boot_time>{{ data.recent_booted_at }}</last_boot_time>
  <vnc_port>{{ data.vnc_port|default_if_none:"" }}</vnc_port>
  <vnc_password>{{ data.vnc_passwd }}</vnc_password>
</vm>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

smigrate (*request*, *format=None*, *vm_uuid=None*)

Migrate the online VM disks from A storage to B storage.

- uri: /vms/:vm_uuid/smigrate/
- method: POST
- **process:**

  - snapshot VM using qmp to dest storage.
  - stream to the snapshot
  - delete VM origin and snapshots
  - update created_at of VM
  - save streamed disks of VM into DB
  - update xen config file

- request:

```
<vm>
  <storlink>2</storlink>
</vm>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

update (*request*, *format=None*, *vm_uuid=None*)

Change VM's cpu, memory spec. online or offline.

- **uri:**

  /vms/:vm_uuid/resize/ (pengx2 POST URL)
  /vms/:vm_uuid/update/ (pengx3 PUT URL)

- method: POST, PUT (PUT is recommended.POST is only used for pengx2)
- request:

```
<vm id='$vm_uuid'>
  <cpu_core>2</cpu_core>
  <memory>1024</memory>
```

```
  <maxmem>2048</maxmem>
</vm>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_204_NO_CONTENT
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

class apiv2.vms.viewsets.**VMViewV2**

edit(*request*, *format=None*, *vm_uuid=None*)
Change VM's name, description (online or offline)

- **uri:**

  /vms/:vm_uuid/edit/

- method: PUT
- request:

```
<vm id='$vm_uuid'>
  <name>test</cpu_core>
  <desc>1024</desc>
</vm>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_204_NO_CONTENT
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

CHAPTER

FOUR

VM CONTROL API

class api.vms.viewsets.**VMControlView**

migrate(*request*, *format=None*, *vm_uuid=None*)

Migrate the VM. (== xl migrate)

- uri: /vms/:vm_uuid/migrate/
- method: POST
- request:

```
<vm>
  <pm_id>$pm_uuid</pm_id>
</vm>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

model

alias of VM

poweroff(*request*, *format=None*, *vm_uuid=None*)

Force to stop the VM.

- uri: /vms/:vm_uuid/poweroff/
- method: POST
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

success: HTTP_200_OK
failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

reboot (*request*, *format=None*, *vm_uuid=None*)
Reboot the VM.

- uri: /vms/:vm_uuid/reboot/
- method: POST
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

success: HTTP_200_OK
failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

reset (*request*, *format=None*, *vm_uuid=None*)
Force to reboot the VM. (== xl destroy and xl create)

- uri: /vms/:vm_uuid/reset/
- method: POST
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

success: HTTP_200_OK
failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

resume (*request*, *format=None*, *vm_uuid=None*)
Resume the VM. (== xl unpause)

- uri: /vms/:vm_uuid/resume/
- method: POST
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

success: HTTP_200_OK

failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

shutdown(*request*, *format=None*, *vm_uuid=None*)
Stop the VM.

- uri: /vms/:vm_uuid/shutdown/
- method: POST
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

start(*request*, *format=None*, *vm_uuid=None*, *snapshot_uuid=None*)
Start VM.

- **uri:**

  /vms/:vm_uuid/start/ (to start the current VM)
  /vms/:vm_uuid/start/snapshots/:snapshot_uuid/ (to start with the specific snapshot VM)

- method: POST
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

suspend(*request*, *format=None*, *vm_uuid=None*)
Suspend the VM. (== xl pause)

- uri: /vms/:vm_uuid/suspend/
- method: POST
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

success: HTTP_200_OK
failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

CHAPTER

FIVE

VM SNAPSHOT API

class api.vms.viewsets.**SnapshotView**

create(*request*, *format=None*, *vm_uuid=None*)

Create a snapshot of VM.

- uri: /vms/:vm_uuid/snapshots/
- method: POST
- request:

```
<snapshot>
  <name>$snapshot_name</name>
  <storlink>$storlink_id</storlink>
  <desc>$snapshot_desc</desc>
</snapshot>
```

- response:

```
<snapshot id="{{ data.uuid }}">
  <name>{{ data.name }}</name>
  <type>{{ data.type }}</type>
  <parent>{{ data.parent }}</parent>
  <date>{{ data.created_at|date:"Y-m-d H:i:s" }}</date>
  <desc>{{ data.desc }}</desc>
</snapshot>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

delete(*request*, *format=None*, *vm_uuid=None*, *snapshot_uuid=None*)

Delete a snapshot of VM.

- uri: /vms/:vm_uuid/snapshots/:snapshot_uuid/
- method: DELETE
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_204_NO_CONTENT
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

index (*request*, *format=None*, *vm_uuid=None*)
List snapshots of VM.

- uri: /vms/:vm_uuid/snapshots/
- method: GET
- request: None
- response:

```
<snapshots>{% for snapshot in data %}
  <snapshot id="{{ snapshot.id }}">
    <name>{{ snapshot.name }}</name>
    <date>{{ snapshot.created_at }}</date>
    <description>{{ snapshot.desc }}</description>
  </snapshot>{% endfor %}
</snapshots>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

model
alias of `Snapshot`

restore (*request*, *format=None*, *vm_uuid=None*, *snapshot_uuid=None*)
Restore VM to the time of the snapshot.

Restore VM from the snapshot is the same process as creating a snapshot using given snapshot as a parent.

- uri: /vms/:vm_uuid/snapshots/:snapshot_uuid/
- method: POST
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

CHAPTER

SIX

VM BACKUP API

class api.vms.viewsets.**BackupView**

backup(*request*, *format=None*, *vm_uuid=None*)

Backup vm image.

- uri: /vms/:vm_uuid/backups/
- method: POST
- request:

```
<vm>
  <backup_name>$backup_name</backup_name>
  <storlink>2</storlink>
  <desc>backup of vm_uuid</desc>
</vm>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

delete(*request*, *format=None*, *vm_uuid=None*, *backup_uuid=None*)

Delete a backup.

- uri: /vms/:vm_uuid/backups/:backup_uuid/
- method: DELETE
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

success: HTTP_204_NO_CONTENT
failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

index(*request*, *format=None*, *vm_uuid=None*)
List all backup instances from DB.

- uri: /vms/:vm_uuid/backups/
- method: GET
- request: None
- response:

```
<vmbackups>{% for backup in data %}
  <vmbackup id="{{ backup.id }}">
    <name>{{ backup.name }}</name>
    <vm id="{{ backup.vm }}"></vm>
    <date>{{ backup.created_at }}</date>
    <size>{{ backup.size }}</size><!-- unit: MB -->
  </vmbackup>{% endfor %}
</vmbackups>
```

- **HTTP codes:**

success: HTTP_200_OK
failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

model
alias of `Backup`

restore(*request*, *format=None*, *vm_uuid=None*, *backup_uuid=None*)
Restore the VM from the backup.

- uri: /vms/:vm_uuid/backups/:backup_uuid/
- method: POST
- request: None
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

success: HTTP_201_CREATED
failure: HTTP_404_NOT_FOUND, HTTP_400_BAD_REQUEST

CHAPTER

SEVEN

VM DISK API

class api.vms.viewsets.**DiskView**

create(*request*, *format=None*, *vm_uuid=None*)

Create a VM disk.

- uri: /vms/:vm_uuid/disks/
- method: POST
- request:

```
<disk>
  <size>1024</size><!-- in MiB -->
  <storlink>$storlink</storlink>
  <desc>$desc</desc>
</disk>
```

- **process:**
  1. get parameters and retrieve vm object from db.
  2. Get currently running VM (origin or snapshot?)
  3. create disk image for currently running VM
  4. execute "xl block-attach" and xl config-update if VM is running.
  5. write meta data to DB.
  6. return response.
- response:

```
<disk id="{{ data.uuid }}">
  <name>{{ data.uuid }}</name>
  <size>{{ data.size }}</size>
  <vm id="{{ data.vm }}"></vm>
  <desc>{{ disk.desc|default_if_none:"" }}</desc>
  <created>{{ disk.created_at }}</created>
  <format>{{ disk.format }}</format>
  <type>{{ disk.type }}</type>
  <storlink>{{ disk.storlink }}</storlink>
</disk>
```

- **HTTP codes:**

  success: HTTP_201_CREATED

failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

delete (*request*, *format=None*, *vm_uuid=None*, *disk_uuid=None*)
Delete disk object.

- uri: /vms/:vm_uuid/disks/:disk_uuid/
- method: DELETE
- request: None
- **process:**
  1. get parameters and retrieve vm object from db.
  2. delete file from filesystem.
  3. delete disk object from db.
  4. return response.
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_204_NO_CONTENT
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

index (*request*, *format=None*, *vm_uuid=None*, *snapshot_uuid=None*, *backup_uuid=None*)
Retrieve disks of VM/Backup/Snapshot from DB.

- **uri:**

  /vms/:vm_uuid/disks/ (VM current disks)
  /vms/:vm_uuid/disks/snapshots/:snapshot_uuid/ (snapshot disks)
  /vms/:vm_uuid/disks/backups/:backup_uuid/ (backup disks)

- method: GET
- request: None
- response:

```
<disks>{% for d in data %}
  <disk id="{{ d.uuid }}">
    <name>{{ d.uuid }}</name>
    <order>{{ d.order }}</order>
    <size>{{ d.size }}</size>{% if d.vm %}
    <vm id="{{ d.vm }}"></vm>{% elif d.snapshot %}
    <snapshot id="{{d.snapshot}}"></snapshot>{% elif d.backup %}
    <backup id="{{ d.backup }}"></backup>{% endif %}
    <desc>{{ d.desc|default_if_none:"" }}</desc>
    <created>{{ d.created_at }}</created>
    <format>{{ d.format }}</format>
    <type>{{ d.type|default_if_none:"" }}</type>
    <storlink>{{ d.storlink }}</storlink>
  </disk>{% endfor %}
</disks>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

model
alias of `Disk`

CHAPTER

EIGHT

VM INTERFACE API

class api.vms.viewsets.**InterfaceView**

create(*request*, *format=None*, *vm_uuid=None*)

Create VM NIC and attach if VM is running.

- uri: /vms/:vm_uuid/interfaces/
- method: POST
- request:

```
<vms>
  <interface_name>xenbr0</interface_name>
</vms>
```

- **process:**
  1. generate mac address.
  2. execute xl network-attach.
  3. write to db.
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

delete(*request*, *format=None*, *vm_uuid=None*, *interface_id=None*)

Delete VM NIC and detach it if VM is running.

- uri: /vms/:vm_uuid/interface/:interface_id/
- method: DELETE
- **process:**
  1. If VM is running, never detach NIC. It's disastrous!
  2. Delete VM NIC in the DB.

- request:

```
<vms>
  <interface_name>xenbr0</interface_name>
</vms>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_204_NO_CONTENT
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

index(*request*, *format=None*, *vm_uuid=None*)

Get the network interface list of VM.

- uri: /vms/:vm_uuid/interfaces/
- method: GET
- request: None
- response:

```
<interfaces>{% for iface in data %}
  <interface id="{{ iface.id }}">
    <mac>{{ iface.mac|default_if_none:"" }}</mac>
    <name>{{ iface.name|default_if_none:"" }}</name>
    <ip>{{ iface.ip|default_if_none:"" }}</ip>
    <vm id="{{ iface.vm }}" />
  </interface>{% endfor %}
</interfaces>
```

- **HTTP codes:**

  success: HTTP_200_OK
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

model

alias of `VMIface`

class apiv2.vms.viewsets.**InterfaceViewV2**

create(*request*, *format=None*, *vm_uuid=None*)

Create VM NIC and attach if VM is running.

- api version : v2
- uri: /vms/:vm_uuid/interfaces/
- method: POST
- request:

```
<vms>
  <interface_name>xenbr0</interface_name>
```

```
  <ipaddress>192.168.0.1</ipaddress>
</vms>
```

- **process:**
  1. generate mac address.
  2. execute xl network-attach.
  3. write to db.
- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_201_CREATED
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

update(*request*, *format=None*, *vm_uuid=None*, *interface_id=None*)
Update VM NIC

- uri: /vms/:vm_uuid/interface/:interface_id/update/
- method: PUT
- **process:**
  1. If VM is running, never update NIC.
  2. Update VM NIC in the DB.
- request:

```
<vms>
  <ipaddress>192.168.0.1</ipaddress>
</vms>
```

- response:

```
<root>
  <result>{{ data.result }}</result>
  <message>{{ data.message }}</message>
</root>
```

- **HTTP codes:**

  success: HTTP_204_NO_CONTENT
  failure: HTTP_400_BAD_REQUEST, HTTP_404_NOT_FOUND

CHAPTER

NINE

MODELS

class pengx.models.**Backup**(**args*, ***kwargs*)
Backup VM model

class pengx.models.**DNS**(*id*, *pm*, *ipaddress1*, *ipaddress2*)

class pengx.models.**Disk**(**args*, ***kwargs*)
VM disk model

path()
Return a full path of vm disk.

class pengx.models.**GVM**(**args*, ***kwargs*)
Golden VM(Template) model

disks4xencfg(*s_iso=None*)
Return disk information of GVM.

- **Arguments:**

  i_snapshot: iso image path.

- **Return:**

  l_disks: list,
  [{'format':,'vdev':,'access':,'target':,'order':},...]

is_windows()
Return boolean if GVM OS is windows or not.

real_vnc_port()
Get a real vnc port from xenstore of VM.

class pengx.models.**ISO**
note: not use db table

class pengx.models.**Network**(**args*, ***kwargs*)
category: nic, bridge, bond method: manual, static, dhcp, none bridge_type: bridge, ovs bridge_ports: comma separated interface names bond_slaves: comma separated interface names

bond_mode_str()
https://www.kernel.org/doc/Documentation/networking/bonding.txt

class pengx.models.**OCFS2**(*id*, *storage*, *pm_uuid*, *peer_ip*, *peer_name*, *number*)

class pengx.models.**PM**(*uuid*, *name*, *cpu*, *memory*, *ssh_pubkey*, *created_at*, *updated_at*)

class pengx.models.**SRIOV**(**args*, ***kwargs*)
pf: pf nic (eth0, eth1) vf: pci bdf (03:10.0, 03:10.1)

class pengx.models.**Snapshot**(**args*, ***kwargs*)
VM Snapshot model

is_parent(*s_uuid*)
Return boolean about whether s_uuid is parent of other snapshots.

Arguments: s_uuid: string, snapshot uuid

Return: b_ret: boolean, True or False

class pengx.models.**Storage**(**args*, ***kwargs*)
pm: physical machine category: local, fc, iscsi, nfs, ocfs2, drbd, pv, vg, lv size, free(unsigned integer): MB uuid: generate uuid for NFS

(optional) mount_point: mount path for local, fc, iscsi, nfs, ocfs2, lv target_ip: target ip address for iSCSI target: target id and path for NFS, target name for iSCSI pv_ids: PV IDs (comma separated) lv_no: number of LV belongs to VG vg_id: VG ID device_id: device to format in OCFS2 drbd_rsc_name: DRBD resource name drbd_disk: local device ID to use for DRBD mirror disk drbd_peer_ip: peer ip address (comma separated)

class pengx.models.**Storlink**(**args*, ***kwargs*)
Storage Link for VM disks.

free_size()
Returns free size of the storlink.

class pengx.models.**VM**(*uuid*, *gvm*, *pm*, *name*, *cpu*, *memory*, *maxmem*, *desc*, *pcivf*, *lock_status*, *vnc_port*, *vnc_passwd*, *created_at*, *recent_booted_at*, *state*, *updated_at*)

disks4vm(*o_cur_vm*, *b_showall=False*)
Return disk information of VM(origin, backup, or snapshot).

- **Arguments:**

  o_cur_vm: object, VM origin or snapshot object
  b_showall: boolean, Show saved snapshot disks, default=False

- **Return:**

  l_disks: list, list of disks in DB.

disks4xencfg(*o_snapshot=None*, *b_res_disks=False*)
Return disk information of VM.

- **Arguments:**

  o_snapshot: object, VM snapshot object, default=None
  b_res_disks: boolean, return l_disks with o_disk, default=False

- **Return:**

  l_disks: list,
  if b_res_disks == False:
  [{'format':,'vdev':,'access':,'target':,'order':},...]
  if b_res_disks == True:
  [<o_disk1>, <o_disk2>, ...]

get_cur_vm()
Get the currently running VM object(can be from origin or snapshot).

- Arguments: None

- **Return:**

  o_cur_vm: object, VM origin or snapshot object.

get_state()
state of vm(xen) - RUNNING, SHUTDOWN, PAUSED..

is_windows()
Return boolean if VM OS is windows or not.

real_vnc_port()
Get a real vnc port from xenstore of VM.

class pengx.models.**VMIface**(**args*, ***kwargs*)
switch: interface name (ex: xenbr0, xenbr1) name: interface of vm - eth0, eth1 method: method of vm's interface - static, dhcp, manual

PYTHON MODULE INDEX

p

INDEX

N

O

P

R

S

U

V