Выключение стенда

Пояснение

В состав коллекции **oVirt Ansible Collection** входит роль **shutdown_env**, с помощью которой можно совершить корректное выключение всего стенда.

В отличии от простого выключения хоста, виртуальные машины завершаются командой **shutdown**, в то время как при выключении хоста происходит терминация ВМ. Также если zVirt развёрнут в режиме **HostedEngine**, то перед выключением стенда будет произведен перевод Менеджера управления в режим глобального обслуживания.

В примерах использования ниже предполагается, что Playbook запускается с **Менеджера управления**, информация о запуске Playbook с внешнего хоста представлена во Введении в автоматизацию работы zVirt с помощью Ansible.

1. Использование роли

1. Создайте playbook со следующим содержимым:

```yaml
---
- name: zVirt shutdown environment
  hosts: localhost
  connection: local
  gather_facts: false

  vars:
    engine_url: https://zvirt.vlab.local/ovirt-engine/api  # Здесь нужно указать FQDN Менеджера управления
    engine_user: admin@internal
    engine_password: admin
    engine_cafile: /etc/pki/ovirt-engine/ca.pem

  roles:
    - ovirt.shutdown_env
```

2. Запустите playbook `ansible-playbook shutdown.yml`

2. Включение HostedEngine

1. После включения стенда, подключитесь к хосту, на котором был установлен Менеджер управления, и выполните команду:

```bash
hosted-engine --vm-status
```

2. После того, как вы убедитесь, что менеджер управления включился, отключите режим обслуживания с помощью команды:

```bash
hosted-engine --set-maintenance --mode=none
```

Импорт OVA-файлов в zVirt с помощью "Ansible"

Пояснение

Коллекция Ansible **ovirt** предоставляет возможность импорта ВМ из OVA-файла. Примеры использования предполагают запуск Playbook с **Менеджера управления**, подробная информацию о запуске Playbook с внешнего хоста представлена во Введении в автоматизацию работы zVirt с помощью Ansible.

1. Пример использования

Ниже представлен пример импорта ВМ из OVA-файла в zVirt:

```yaml
---
- name: OVA import
  hosts: localhost
  connection: local
  gather_facts: false

  tasks:
    - name: Login
      ovirt_auth:
        url: "https://uruk-zvirtsa-srv.uruk.local/ovirt-engine/api"
        username: "ansible@internal"
        password: "password"
      register: loggedin
    - name: Import
      ovirt_vm:
        auth: "{{ ovirt_auth }}"
        cluster: Default
        name: Imported_VM
        host: zvirt-host.domain.local
        timeout: 1800
        poll_interval: 30
        kvm:
          name: Imported_VM
          url: ova:///ova/import_vm.ova
          storage_domain: Default
    - name: Logout
      ovirt_auth:
        ovirt_auth: "{{ ovirt_auth }}"
```

```
        state: absent
      when: not loggedin.skipped | default(false)
```

Путь до OVA указывается в блоке **kvm: url**. На директорию и сам OVA-файл необходимо назначить владельца vdsm:kvm, для этого необходимо выполнить команду:

```
chown 36:36 /ova/import_vm.ova
```

Введение в автоматизацию работы zVirt с помощью Ansible

1. Запуск Playbook с Менеджера управления.

на Менеджере управления Ansible используется для контроля и автоматизации процессов zVirt, обновлять **ansible** или **ansible-core** не рекомендуется.

1.1. Подготовка к работе

Для взаимодействия с zVirt с помощью Ansible, необходимо установить коллекцию **ovirt.ovirt**. Установить коллекцию можно с помощью команды

```
ansible-galaxy collection install ovirt.ovirt
```

1.2. Аутентификация в zVirt

Для исполнения модулей **ovirt.ovirt**, необходимо в **Роли Ansible** или **Playbook** авторизоваться в zVirt с помощью модуля **ovirt_auth**. В данном модуле используются URL-адрес API менеджера управления zVirt, **имя пользователя** и **пароль**.

- имя пользователя задаётся полностью, вместе с пространством имён. Внутренние учётные записи имеют пространство имён **internal**, разделителем имени пользователя и пространства имён служит символ @.
- использовать пользователя **admin не рекомендуется**. Рекомендуется создать отдельного пользователя с правами **superuser** и защищённым паролем.

```yaml
- name: Login
  ovirt_auth:
    url: "https://zvirt.example.ru/ovirt-engine/api"
    username: "ansible@internal"
    password: "password"
```

Для zVirt версии 4.2 и выше, если вместо AAA JDBC используется Keycloak, необходимо использовать пространство имен **internalsso**.

Данный модуль задаёт переменную **auth**, которая в дальнейшем вызывается другими модулями. Для вызова данной переменной при выполнении **task**, необходимо указать её в самом модуле. Ниже представлен пример с модулем **ovirt_vm**:

```yaml
- name: Create VM
  ovirt_vm:
    auth: "{{ ovirt_auth }}"
    state: present
    name: VM_Name
    cluster: Cluster_Name
```

1.3. Сброс токена аутентификации

Важно отметить, что если запуск Playbook осуществляется с использованием пользователя, для которого ограничено количество одновременных сессий, то необходимо в конце Playbook добавить Task, который завершит сеанс по завершению Playbook, иначе пользовательская сессия останется активна, что будет препятствовать последующим запускам Ansible Playbook. Лучшей практикой написания Playbook является пример ниже:

```yaml
  tasks:
  - name: Login
    ovirt_auth:
      url: "https://zvirt.example.ru/ovirt-engine/api"
      username: "ansible@internal"
      password: "password"
    register: loggedin

  - task1
  - task2
  - task3

  - name: Logout from oVirt
    ovirt_auth:
      state: absent
      ovirt_auth: "{{ ovirt_auth }}"
    when: not loggedin.skipped | default(false)
```

В данном примере сначала выполняется аутентификация, затем выполняются рабочие Task'и.

При выполнении **Login** Task'а регистрируется переменная **loggedin**. После всех задач следуют Task, удаляющий токен аутентификации.

Принцип работы следующий: используется полученный ранее в качестве переменной токен аутентификации и ему придаётся статус **absent**, что и является удалением токена. У

Task'и также используется условие, проверяющее была ли аутентификация через переменную **loggedin**.

2. Запуск Playbook с внешнего хоста.

2.1. Установка Ansible

Установка с помощью менеджера пакетов:

Ansible входит в репозиторий **epel-release**, перед установкой Ansible нужно установить репозиторий, это можно сделать командой:

```bash
dnf install -y epel-release
```

После успешной установки `epel-release`, можно установить Ansible с помощью команды:

```bash
dnf install -y ansible
```

Установка с помощью Python pip:

1. Команда для установки с Python может отличаться в зависимости от версии: **Python 2** или **Python 3**. Версию **Python 2** можно узнать с помощью команды `python --version`.

   Если установлен **Python 3**, то версию можно узнать командой `python3 --version`.

2. Чтобы установить Ansible с помощью Python модуля `pip`
   - В случае с `Python 2`, нужно ввести команду:

     ```bash
     pip install ansible
     ```

     Либо

     ```bash
     python -m pip install ansible
     ```

   - В случае с `Python 3`, необходимо ввести команду:

     ```bash
     pip3 install ansible
     ```

     Либо

     ```bash
     python3 -m pip install ansible
     ```

При работе с **pip**, рекомендуется добавлять ключ **--user** при установке от пользователя, не являющегося **root**, чтобы установленный Ansible можно было запускать под этим пользователем.

1. Проверить корректно ли установился **ansible** можно с помощью команды:

```bash
ansible --version
```

Требования:

1. Python >= 2.7 ИЛИ Python >= 3.6
2. ovirt-engine-sdk-python

2.2. Установка коллекции

Взаимодействие с zVirt осуществляется с помощью модулей коллекции **ovirt.ovirt**.

Если Ansible устанавливался как пакет **rpm/deb** или с помощью системы управления пакетами Python **pip**, данная коллекция может быть предустановлена.

Проверить список установленных коллекций можно с помощью команды `ansible-galaxy collection list`.

Если среди установленных коллекций отсутствует `ovirt.ovirt`, установить её можно с помощью команды `ansible-galaxy collection install ovirt.ovirt`.

Чтобы использовать модуль в собственных `playbook`, нужно указать в качестве task `ovirt.ovirt.ovirt_vm`

2.3. Аутентификация в zVirt

Для исполнения модулей **ovirt.ovirt**, необходимо в Роли *Ansible* или *Playbook* авторизоваться в zVirt с помощью модуля **ovirt_auth**. В данном модуле используются URL-адрес API менеджера управления zVirt, **имя пользователя** и **пароль**.

Имя пользователя задаётся полностью, включая пространство имён. Внутренние учётные записи имеют пространство имён **internal**, разделителем имени пользователя и пространства имён служит символ @.

Использовать пользователя **admin не рекомендуется**. Рекомендуется создать отдельного пользователя с правами **superuser** и защищённым паролем.

```yaml
- name: Login
  ovirt_auth:
    url: "https://zvirt.example.ru/ovirt-engine/api"
    username: "ansible@internal"
    password: "password"
```

Данный модуль задаёт переменную **auth**, которая в дальнейшем вызывается другими модулями. Для вызова данной переменной при выполнении **task**, необходимо указать её в самом модуле. Ниже представлен пример с модулем **ovirt_vm**:

```yaml
- name: Create VM
  ovirt_vm:
    auth: "{{ ovirt_auth }}"
    state: present
    name: VM_Name
    cluster: Cluster_Name
```

Все модули и их вызовы описаны в официальном руководстве Ansible

2.4. Сброс токена аутентификации

Важно отметить, что если запуск Playbook осуществляется с использованием пользователя, для которого ограничено количество одновременных сессий, то необходимо в конце Playbook добавить Task, который завершит сеанс по завершению Playbook, иначе пользовательская сессия останется активна, что будет препятствовать последующим запускам Ansible Playbook.

Лучшей практикой написания Playbook является пример ниже:

```yaml
  tasks:
  - name: Login
    ovirt_auth:
      url: "https://zvirt.example.ru/ovirt-engine/api"
      username: "ansible@internal"
      password: "password"
    register: loggedin

  - task1
  - task2
  - task3

  - name: Logout from oVirt
    ovirt_auth:
```

```
      state: absent
      ovirt_auth: "{{ ovirt_auth }}"
    when: not loggedin.skipped | default(false)
```

В данном примере сначала выполняется аутентификация, затем выполняются рабочие Task'и.

При выполнении **Login** Task'а регистрируется переменная `loggedin`. После всех задач следуют Task, удаляющий токен аутентификации.

Принцип работы следующий: используется полученный ранее в качестве переменной токен аутентификации и ему придаётся статус **absent**, что и является удалением токена. У Task'и также используется условие, проверяющее была ли аутентификация через переменную **loggedin**.

2.5. Запуск Playbook

При запуске Playbook с внешнего хоста, в качестве хоста **Play** нужно указать **localhost**, так как дальнейшие операции со zVirt будут проводиться при подключении к API zVirt.

Playbook ниже является корректным примером:

YAML

```yaml
---
- name: zVirt Playbook
  hosts: localhost
  connection: local
  gather_facts: false

  tasks:
    - name: Loggin
      ovirt_auth:
        url: "https://zvirt.example.ru/ovirt-engine/api"
        username: "ansible@internal"
        password: "password"
    - name: Second Task
    - name: Third Task
```

запуск **Playbook** с Task'ами для zVirt настоятельно не рекомендуется с указанием хоста, отличного от **localhost**, и тем более группы хостов. При запуске **Playbook** с указанием иного хоста в лучшем случае может завершиться ошибкой, если не установлены необходимые SDK. Если же SDK установлены и хостов в группе 2 то, например, Task будет запущен и отправлена команда в API с каждого из хостов, что может привести к проблемам.

3. Коллекция Ovirt Ansible Collection

Для работы с zVirt можно использовать коллекцию oVirt Ansible Collection, доступная для скачивания в архиве.

В данную коллекцию включены несколько шаблонов ролей, примеров работы с коллекцией **ovirt.ovirt**, а также плагинов.

Чтобы начать пользоваться ролями из данной коллекции, необходимо копировать каталог **roles** из архива на хост, с которого будет запускаться **playbook**, и включить роль в **playbook**, как на примере ниже:

```yaml
---
- name: Test Playbook
  hosts: Host
  become: true

  roles:
    - engine_setup
```

3.1. Описание ролей

disaster_recovery: данная роль позволяет управлять сценариями аварийного восстановления;

- **image_template**: данная роль позволяет создавать шаблоны из внешних образов. Пример применения в инструкции по работе с шаблонами с помощью Ansible.
- **infra**: данная роль позволяет настраивать инфраструктуру zVirt, в том числе пул mac-адресов, центры данных, кластеры, сети, хосты, пользователей и группы. Пример применения в инструкции по управлению инфраструктурой кластера с помощью Ansible.
- **remove_stale_lun**: данная роль итерируется на всех хостах в центре данных и удаляет устаревшие LUN-устройства со всех хостов. Пример применения в инструкции по удалению устаревших LUN на хостах средствами Ansible.
- **shutdown_env**: данная роль выключает весь стенд zVirt. Пример применения в инструкции по выключению стенда с помощью Ansible.
- **vm_infra**: данная роль позволяет управлять инфраструктурой виртуальных машин в zVirt. Пример применения в инструкции по управлению инфраструктурой виртуальных машин с помощью Ansible.

Управление инфраструктурой кластера с помощью Ansible

Пояснение

Для управления не виртуальной инфраструктурой кластера, для Ansible существует роль **infra**, входящая в состав **oVirt Ansible Collection**.

С помощью данной роли можно управлять следующими сущностями кластера:

- центрами данных
- пулами MAC-адресов
- кластерами и профилями кластеров
- хостами
- сетями
- доменами хранения
- пользователями и правами
- внешними провайдерами

Также для управления инфраструктурой кластера можно управлять с помощью модулей Ansible коллекции **ovirt.ovirt**.

В примерах использования ниже предполагается, что Playbook запускается с **Менеджера управления**, информация о запуске Playbook с внешнего хоста представлена во Введении в автоматизацию работы zVirt с помощью Ansible.

1. Использование роли

Инфраструктура, которая необходима в качестве результата, при использовании роли, в Playbook описывается в качестве переменных, которые далее передаются роли. Все переменные можно посмотреть в инструкции **README.md**, которая находится в каталоге с ролью.

Обязательными являются переменные, необходимые для авторизации в zVirt:

YAML

```yaml
vars:
  engine_fqdn: ovirt-engine.example.com
  engine_user: ansible@internal
```

```
  engine_password: 123456
  engine_cafile: /etc/pki/ovirt-engine/ca.pem
```

2. Управление кластером

В кластере можно настроить свойства кластера, в том числе **Memory Ballooning**, **тип CPU**, **политики оптимизации памяти**, **политики миграции** и так далее. С помощью роли можно задать 1 из 2 преднастроенных профилей кластера: **production** и **development**.

Различия профилей представлены ниже

```
Production
| Parameter                          | Value              |
|------------------------------------|--------------------|
| ballooning                         | false              |
| ksm                                | false              |
| host_reason                        | true               |
| vm_reason                          | true               |
| memory_policy                      | disabled           |
| migration_policy                   | suspend_workload   |
| scheduling_policy                  | evenly_distributed |
| ha_reservation                     | true               |
| fence_enabled                      | true               |
| fence_skip_if_connectivity_broken  | true               |
| fence_skip_if_sd_active            | true               |

Development
| Parameter         | Value          |
|-------------------|----------------|
| ballooning        | true           |
| ksm               | true           |
| host_reason       | false          |
| vm_reason         | false          |
| memory_policy     | server         |
| migration_policy  | post_copy      |
```

Переменные для управления кластером необходимо указывать в виде списка, в котором перечислены кластеры и их параметры:

YAML

```yaml
vars:
  clusters:
    - name: Cluster1
      cpu_type: Inter Conroe Family
      profile: production
    - name: Cluster2
      state: present
      cpu_type: AMD Opteron G3
```

```
      ballooning: true
      memory_policy: server
```

3. Управление хостами

Переменные для хостов можно перечислить в отдельном файле, либо через список **hosts**.

Если хосты перечислены в отдельном файле, в Playbook это можно указать так:

YAML

```yaml
vars:
  hosts_var_name: ovirt_hosts
```

Либо можно перечислить в виде переменных в самом Playbook:

YAML

```yaml
vars:
  hosts:
    - name: Host1
      address: 1.2.3.4
      cluster: Default
      password: 12345
      hosted_engine: deploy         # Хост будет подготовлен для размещения HE
    - name: Host2
      address: 1.2.3.5
      cluster: production
      password: 12345
      state: present                # Хост будет добавлен в кластер, значение
present используется по умолчанию
      hosted_engine: undeploy       # Хост будет перенастроен таким образом,
чтобы Hosted Engine мог быть размещён на данном хосте
    - name: Host3
      address: 1.2.3.6
      cluster: development
      password: 12345
      state: absent                 # Хост будет удалён из кластера
```

Список хостов в отдельном файле перечисляется также, как в Playbook и должен иметь такой же вид.

4. Управление доменами хранения

Для управления доменами хранения, перечислите их в списке **storages**

YAML

```yaml
vars:
  storages:
    nfs_storage:
```

```
      master: true                    # Указывает мастер домен, при добавлении
нескольких доменов сразу,
                                      # параметр master укажет какой необходимо
добавить первым
      state: present
      nfs:
        address: 10.11.12.13
        path: /path/to/storage
    iscsi_storage:
      state: present
      iscsi:
        target: iqn.2023-04.zvirt.ru:storage
        port: 3260
        address: 10.11.12.14
        username: user
        password: password
        lun_id: LUN ID
    export_storage:
      domain_function: export
      nfs:
        address: 10.11.12.15
        path: /path/to/storage-export
    iso_storage:
      domain_function: iso
      nfs:
        address: 10.11.12.16
        path: /path/to/storage-iso
```

5. Управление сетями

Управления сетями можно разделить на управление логическими сетями кластера и управление сетями хостов.

YAML

```
# Логические сети
vars:
  logical_networks:
    - name: mynetwork
      clusters:
        - name: Default
          assigned: yes
          required: no
          display: no
          migration: yes
          gluster: no
# Сети хостов
  host_networks:
    - name: host1
      check: true
```

```
      save: true
      bond:
        name: bond0
        mode: 2
        interfaces:
          - eth2
          - eth3
      networks:
        - name: mynetwork
          boot_protocol: dhcp
```

Параметр **check** и **save** соответствуют флажкам
[Проверить соединение между хостом и Engine] и
[Проверить соединение между хостом и Engine], описание которых можно посмотреть в настройке сетей хоста.

6. Управление пользователями

Управление пользователями, группами и правами имеет следующий вид:

YAML

```yaml
vars:
 users:
  - name: first.user
    authz_name: internal-authz
    password: 123456
    valid_to: "2018-01-01 00:00:00Z"
  - name: second.user
    authz_name: internal-authz
    password: 123456
    valid_to: "2018-01-01 00:00:00Z"
  - name: third.user
    authz_name: internal-authz
    state: absent                              # Удаление пользователя
 user_groups:
  - name: admins
    authz_name: internal-authz
    users:
     - first.user
     - second.user
 permissions:
  - state: present
    user_name: first.user
    authz_name: internal-authz
    role: SuperUser
    object_type: cluster
    object_name: production
  - state: present
    group_name: admins
```

```
    authz_name: internal-authz
    role: UserVmManager
    object_type: cluster
    object_name: development
```

Управление инфраструктурой виртуальных машин с помощью Ansible

Пояснение

В состав **oVirt Ansible Collection** входит роль **vm_infra**. Данная роль позволяет управлять инфраструктурой виртуальных машин в zVirt. Также данная роль также позволяет вести **Inventory-файл** созданных машин и группировать их на основе тэгов.

Примеры использования предполагают запуск **Playbook** с **Менеджера управления**, подробная информацию о запуске **Playbook** с внешнего хоста представлена во Введении в автоматизацию работы zVirt с помощью Ansible.

1. Использование

1.1. Общая структура

Для работы с данной ролью нужно использовать определённую структуру Playbook. Данная структура состоит из 3 условных блоков:

1. переменные, описывающие профиль виртуальной машины
2. переменная, состоящая из составного списка
3. запуск роли

Создание ВМ в данной роли с использованием профилей является лучшей практикой.

Пример Playbook представлен ниже:

YAML

```yaml
- name: zVirt VM Infra
  hosts: localhost
  connection: local
  gather_facts: false

  vars:
    engine_fqdn: zvirt-engine.example.ru
    engine_user: ansible@internal
    engine_password: ansible
    engine_cafile: /etc/pki/ovirt-engine/ca.pem

    # Ниже указывается профиль для виртуальных машин, которые в дальнейшем будут
```

```
выполнять роль веб-сервера
    httpd_vm:
      cluster: some_cluster
      # Параметр domain ниже отвечает за домен виртуальной машины, а не домен
хранения
      domain: example.ru
      template: some_template
      memory: 2GiB
      # Количество CPU задаётся параметром cores
      cores: 2
      # Под параметром disks задаются виртуальные диски ВМ
      disks:
        - size: 20GiB
          name: data
          storage_domain: some_storage_domain
          interface: virtio
      # Под параметром nics задаются сетевые интерфейсы
      nics:
        - name: ovirtmgmt
          network: ovirtmgmt
          profile: ovirtmgmt

    # Далее указывается профиль для виртуальных машин, которые будут выполнять
роль базы данных
    db_vm:
      cluster: some_cluster
      domain: example.ru
      template: some_template
      memory: 4GiB
      cores: 1
      disks:
        - size: 50GiB
          name: data
          storage_domain: some_storage_domain
          interface: virtio
      nics:
        - name: ovirtmgmt
          network: ovirtmgmt
          profile: ovirtmgmt

    # В списке vms указывается список самих виртуальных машин, которые будут
созданы во время работы роли
    vms:
      - name: postgresql-vm-0
        tag: database-servers
        profile: "{{ db_vm }}"
      - name: postgresql-vm-1
        tag: database-servers
        profile: "{{ db_vm }}"
      - name: apache-vm
        tag: web-servers
```

```
        profile: "{{ httpd_vm }}"

  roles:
    - vm_infra
```

Управления разрешениями на виртуальные машины zVirt с помощью Ansible

Пояснение

При необходимости назначение разных прав на большое количество виртуальных машин или иных ресурсов zVirt, эту задачу можно автоматизировать с помощью Ansible.

Для работы с разрешениями в коллекции Ansible **ovirt.ovirt** существует модуль **ovirt_permission**.

С помощью данного модуля можно управлять правами следующих ресурсов:

- центры данных
- кластеры
- хосты
- сети
- домены хранения
- диски
- ВМ и их шаблонами

И некоторыми другими сущностями в zVirt. Полную информацию можно получить в официальной документации Ansible на коллекцию **ovirt.ovirt`** и модуль **ovirt_permission** в частности.

В примерах использования ниже предполагается, что Playbook запускается с **Менеджера управления**, информация о запуске Playbook с внешнего хоста представлена во Введении в автоматизацию работы zVirt с помощью Ansible.

1. Пример использования: назначение прав на ВМ

Чтобы назначить права на виртуальные машины с помощью Ansible, необходимо составить **playbook** следующего вида:

```yaml
- name:
  hosts: localhost
```

```
  connection: local
  gather_facts: false

  tasks:
    - name: login
      ovirt_auth:
        url: "https://zvirt.example.ru/ovirt-engine/api"
        username: "ansible@internal"
        password: "password"
      register: loggedin
    - name: Change Permissions
      ovirt_permission:
         auth: "{{ ovirt_auth }}"
         state: present                                    # Указывает
должны разрешение присутствовать или отсутствовать, по умолчанию используется
значение present. Если права необходимо убрать, нужно заменить present на absent
         user_name: myuser                                 # Если будет
указан параметр user_name, то права будут выданы данному пользователю
         # group_name: mygroup                             # Либо можно
указать параметр group_name и выдать права на ресурс группе
         authz_name: internal-authz                        # В данном
параметре указывается провайдер авторизации. По умолчанию провайдером
авторизации для локальных пользователей является internal-authz
         # authz_name: zvirt.domain.ru                     # Если
необходимо назначить права для доменной группы, нужно указать ваш домен
         object_type: vm                                   # В параметре
object_type указывает тип объекта, например, ВМ, домен хранения или хост
         object_name: "{{ item }}"                         # В object_name
задаётся имя объекта и при необходимо выдать права на большое количество
объектов можно воспользоваться циклом, передав список названий прямо в Playbook
или внешним файлом
         # object_id: ** ID **                             # Можно
использовать вместо object_name, позволяет указывать ID объектов вместо имён
         role: SuperUser                                   # Здесь
указывается какую роль задать при добавлении пользователя / группы в разрешения
на объект
      loop:
        - VM1
        - VM2
    - name: Logout
      ovirt_auth:
        ovirt_auth: "{{ ovirt_auth }}"
        state: absent
      when: not loggedin.skipped | default(false)
```

В примере выше на виртуальные машины **VM1** и **VM2** выдаются права пользователю локальному **myuser**. По завершению выполнения Playbook, у пользователя будут права роли **SuperUser** на данные виртуальные машины.

Руководство по REST API для SDN

Версия zVirt: 4.2

1. Введение

Для поддержки и управления структурой программно-управляемых сетей посредством архитектуры REST предоставляется API для SDN на базе сервиса 'zvirt-engine-backend'.

Данное API дополняет оригинальное REST API zVirt и придерживается основных принципов его построения, но в отличии от оригинального реализует доступ по HTTP(S) протоколу, без реализации SDK.

Эта документация служит справочником по REST API для SDN. Она предназначена для предоставления разработчикам и администраторам инструкций и примеров, которые помогут использовать функциональные возможности программно-управляемых сетей через API.

2. Предварительные требования к API

Предварительные требования для использования REST API для SDN:

- Сетевая установка zVirt Engine, включающая API.
- Клиент или программная библиотека, которая инициирует и получает HTTP-запросы от сервера API. Например:
  - Инструмент командной строки cURL.
  - RESTClient — отладчик для веб-служб RESTful.
  - Postman — HTTP-клиент для тестирования API.
- Знание протокола передачи гипертекста (HTTP), протокола, используемого для взаимодействия с REST API. Инженерная рабочая группа Интернета публикует запрос комментариев (RFC) с объяснением протокола передачи гипертекста по адресу http://www.ietf.org/rfc/rfc2616.txt.
- Знание расширяемого языка разметки (XML) или нотации объектов JavaScript (JSON), которые API использует для создания представлений ресурсов. W3C предоставляет полную спецификацию XML на http://www.w3.org/TR/xml. ECMA International предоставляет бесплатную публикацию по JSON на http://www.ecma-international.org.

3. Аутентификация и безопасность

Предпочтительным механизмом аутентификации является OAuth 2.0, как описано в RFC 6749.

OAuth — это сложный протокол с несколькими механизмами получения токенов авторизации и доступа. Для использования с API поддерживается только предоставление учетных данных владельца ресурса, как описано в разделе 4.3 RFC 6749.

Сначала необходимо получить **токен**, отправив имя пользователя и пароль в службу единого входа Менеджера управления:

```
POST /ovirt-engine/sso/oauth/token HTTP/1.1
Host: myengine.example.com
Content-Type: application/x-www-form-urlencoded
Accept: application/json
```

Тело запроса должно содержать параметры `grant_type`, `scope`, `username` и `password`:

Таблица 1. Параметры запроса токена OAuth

Имя	Значение
grant_type	password
scope	ovirt-app-api
username	admin@internal
password	mypassword

Эти параметры должны быть представлены в URL кодировке. Например, символ `@` в имени пользователя должен быть закодирован как `%40`. Результирующее тело запроса будет примерно таким:

```
grant_type=password&scope=ovirt-app-
api&username=admin%40internal&password=mypassword
```

Параметр `scope` описан как необязательный в OAuth RFC, но при использовании его с API он является обязательным, и его значение должно быть `ovirt-app-api`.

Если имя пользователя и пароль верны, служба единого входа Менеджера управления ответит документом JSON, подобным этому:

```
{
  "access_token": "fqbR1ftzh8wBCviLxJcYuV5oSDI=",
  "token_type": "bearer",
  "scope": "...",
  ...
}
```

Для целей аутентификации API единственной подходящей парой имя/значение является `access_token`. Ни в коем случае не манипулируйте этим; используйте его точно так, как это предусмотрено службой единого входа.

После получения токена его можно использовать для выполнения запросов к API, включив его в заголовок HTTP `Authorization` и используя схему `Bearer`. Например, чтобы получить список виртуальных машин, отправьте такой запрос:

```
GET /ovirt-engine/api/vms HTTP/1.1
Host: myengine.example.com
Accept: application/xml
Authorization: Bearer fqbR1ftzh8wBCviLxJcYuV5oSDI=
```

Токен можно использовать несколько раз для нескольких запросов, но в конечном итоге срок его действия истечет. По истечении этого срока сервер отклонит запрос с кодом ответа HTTP **401**:

```
HTTP/1.1 401 Unauthorized
```

В этом случае требуется новый токен, так как служба единого входа Менеджера управления в настоящее время не поддерживает обновление токенов. Новый токен можно запросить тем же способом, который описан выше.

3.1. Ролевая модель

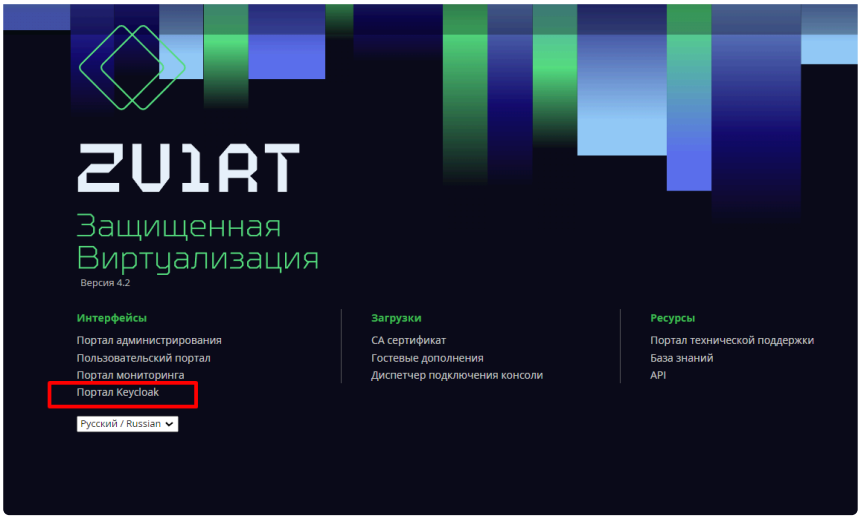
При обращении к REST API для SDN от имени пользователя необходимо учитывать, что у пользователя должны быть роли SDN Manager, SDN ACL Manager, со следующими привилегиями:

- SDN_INFRASTRUCTURE_MANAGEMENT, используется для всех запросов на создание, редактирование и удаление логических сетей, внешних сетей, подсетей, маршрутизаторов и интерфейсов маршрутизатора;
- SDN_ROUTER_CONFIGURATION используется для запросов, редактирующих NAT, таблицу маршрутов и внешний интерфейс маршрутизаторов;
- SDN_ACL_MANAGEMENT используется для запросов, связанных с созданием, редактированием и удалением групп безопасности и правил безопасности;

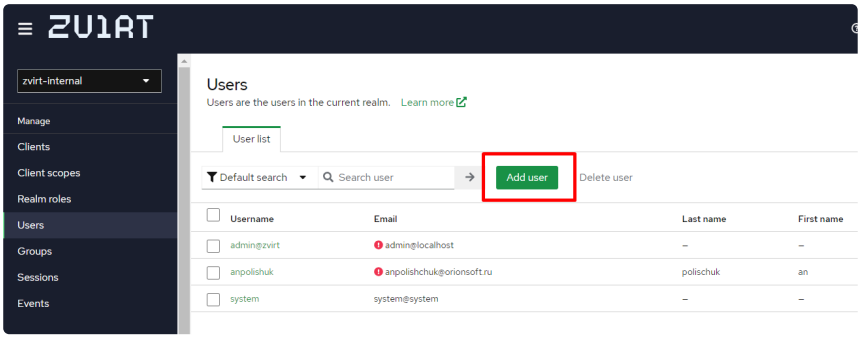
- SDN_ACL_CONFIGURATION используется для запросов, связанных с добавлением/удалением групп безопасности на портах, а также с управлением политикой безопасности портов.

Для добавления необходимых ролей и привилегий пользователю необходимо:

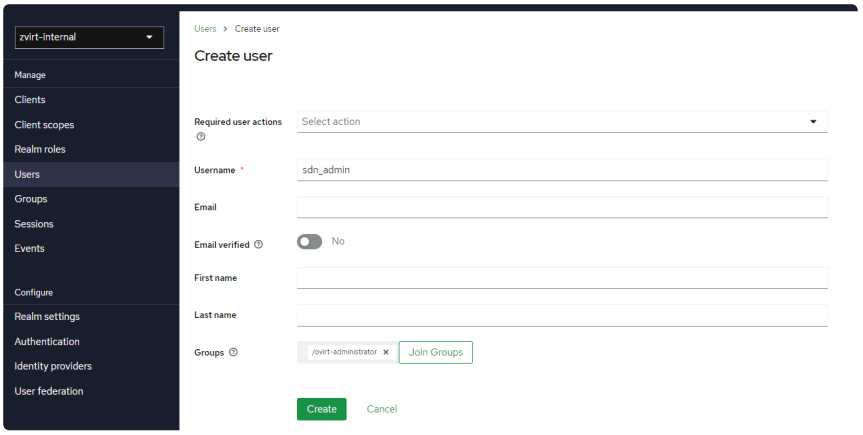
- Создать пользователя через **Портал Keycloak**, для этого:
  - Подключитетесь к порталу Keycloak;



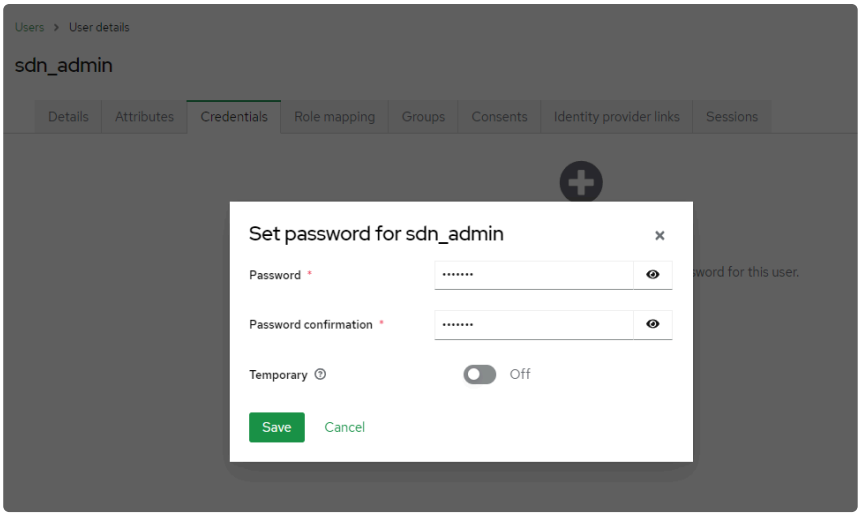
  - Перейдите во вкладку **Users** и нажмите **[Add User]**;



  - Задайте пользователю имя в поле **Username** и добавьте в группу **ovirt-administrator** в поле **Groups**;

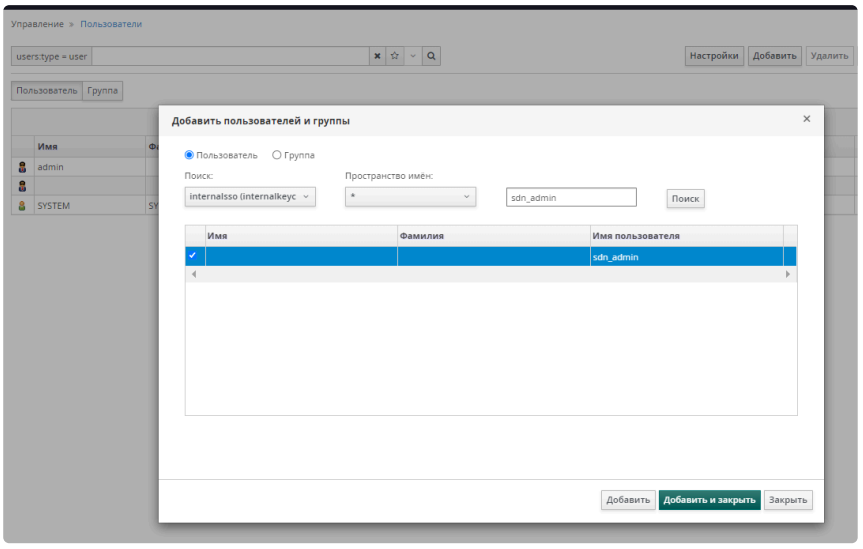


  - После создания пользователя, перейдите в его свойства и во вкладке **Credentials** задайте пароль нажав на **[Set password]**;

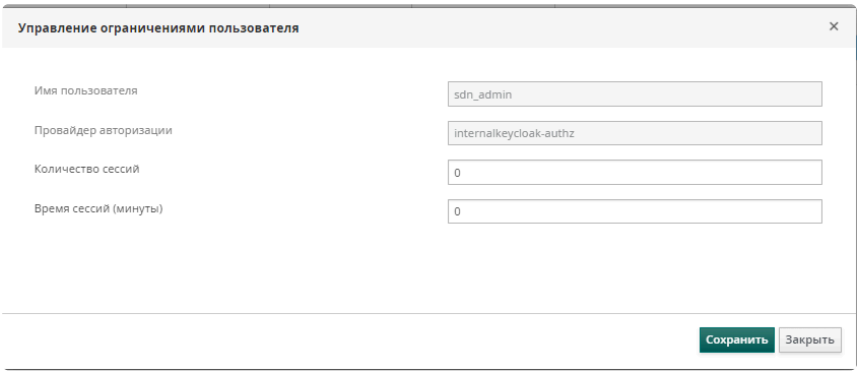


- Настроить роли и привилегии пользователя, для этого:
  - На Портале администрирования перейдите в **Управление › Пользователи**.

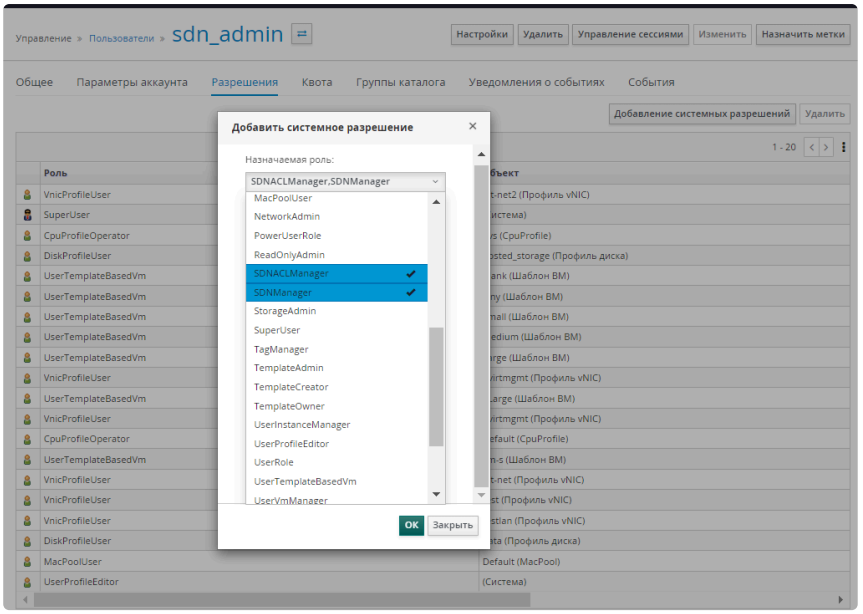
    Нажмите **[Добавить]**, в параметрах поиска в качестве базы пользователей укажите **internalsso (internalkeycloak-authz)** и введите имя пользователя в поисковой строке. После обнаружения пользователя, выделите его в списке и нажмите **[Добавить и закрыть]**.



  - Для снятия ограничений по количеству сессий и времени сессии, выделите пользователя из списка нажмите **[Управление сессиями]** и в полях **Количество сессий** и **Время сессий(минуты)** установите значение **0**;



- Для добавления привилегий пользователю необходимо открыть свойства пользователя, далее во вкладке **Разрешения** нажать **[Добавление системных разрешений]** выбрать из списка роли **SDNACLManager** и **SDNManager**.



3.2. TLS/SSL-сертификация

API zVirt требует защищенного протокола передачи гипертекста (HTTPS) для безопасного взаимодействия с клиентским программным обеспечением. Это включает в себя получение сертификата ЦС, используемого сервером, и его импорт в хранилище сертификатов вашего клиента.

3.2.1. Получение CA сертификата

Вы можете получить CA сертификат от Менеджера управления zVirt и передать его на клиентскую машину одним из следующих способов:

Способ 1

Предпочтительный метод получения CA сертификата — использовать инструмент командной строки `openssl s_client` для выполнения реального рукопожатия TLS с сервером, а затем извлечь сертификаты, которые он представляет. Запустите команду следующим образом:

```
$ openssl s_client \
-connect myengine.example.com:443 \
-showcerts \
< /dev/null
```

Эта команда подключится к серверу и отобразит вывод, подобный следующему:

```
CONNECTED(00000003)
depth=1 C = US, O = Example Inc., CN = myengine.example.com.23416
verify error:num=19:self signed certificate in certificate chain
---
Certificate chain
 0 s:/C=US/O=Example Inc./CN=myengine.example.com
   i:/C=US/O=Example Inc./CN=myengine.example.com.23416
-----BEGIN CERTIFICATE-----
MIIEaTCCA1GgAwIBAgICEAQwDQYJKoZIhvcNAQEFBQAwSTELMAkGA1UEBhMCVVMx
FTATBgNVBAoTDEV4YW1wbGUgSW5jLjEjMCEGA1UEAxMaZW5naW5lNDEuZXhhbXBs
SVlJe7e5FTEtHJGTAeWWM6dGbsFhip5VXM0gfqg=
-----END CERTIFICATE-----
 1 s:/C=US/O=Example Inc./CN=myengine.example.com.23416
   i:/C=US/O=Example Inc./CN=myengine.example.com.23416
-----BEGIN CERTIFICATE-----
MIIDxjCCAq6gAwIBAgICEAAwDQYJKoZIhvcNAQEFBQAwSTELMAkGA1UEBhMCVVMx
FTATBgNVBAoTDEV4YW1wbGUgSW5jLjEjMCEGA1UEAxMaZW5naW5lNDEuZXhhbXBs
Pkyg1rQHR6ebGQ==
-----END CERTIFICATE-----
```

Текст между маркерами `-----BEGIN CERTIFICATE-----` и `-----END CERTIFICATE-----` показывает сертификаты, представленные сервером. Первый — это сертификат самого сервера, а последний — CA сертификат. Скопируйте CA сертификат вместе с метками в файл **ca.crt**. Результат должен выглядеть так:

```
-----BEGIN CERTIFICATE-----
MIIDxjCCAq6gAwIBAgICEAAwDQYJKoZIhvcNAQEFBQAwSTELMAkGA1UEBhMCVVMx
FTATBgNVBAoTDEV4YW1wbGUgSW5jLjEjMCEGA1UEAxMaZW5naW5lNDEuZXhhbXBs
Pkyg1rQHR6ebGQ==
-----END CERTIFICATE-----
```

Это самый надежный способ получить CA сертификат, используемый сервером. Остальные описанные здесь способы сработают в большинстве случаев, но они не дадут правильный CA сертификат, если он был вручную заменен администратором сервера.

Способ 2

Если вы не можете использовать описанный выше метод `openssl s_client`, вы можете вместо этого использовать инструмент командной строки для загрузки CA сертификата из Менеджера управления zVirt.

Примеры инструментов командной строки включают `curl` и `wget`, оба из которых доступны на нескольких платформах.

При использовании `curl`:

```
$ curl \
--output ca.crt \
'http://myengine.example.com/ovirt-engine/services/pki-resource?resource=ca-certificate&format=X509-PEM-CA'
```

При использовании wget:

```
$ wget \
--output-document ca.crt \
'http://myengine.example.com/ovirt-engine/services/pki-resource?resource=ca-certificate&format=X509-PEM-CA'
```

Способ 3

С помощью веб-браузера перейдите к сертификату, расположенному по адресу:

```
https://myengine.example.com/ovirt-engine/services/pki-resource?resource=ca-certificate&format=X509-PEM-CA
```

В зависимости от выбранного браузера сертификат загружается или импортируется в хранилище ключей браузера.

- Если браузер загружает сертификат: сохраните файл как **ca.crt**.
- Если браузер импортирует сертификат: экспортируйте его из параметров сертификации браузера и сохраните как **ca.crt**.

Способ 4

Войдите в Менеджер управления, экспортируйте сертификат из хранилища доверенных сертификатов и скопируйте его на свой клиентский компьютер.

1. Войдите на машину с Менеджером управления как *root*.
2. Экспортируйте сертификат из хранилища доверенных сертификатов с помощью утилиты `keytool`:

   ```
   keytool \
   -keystore /etc/pki/ovirt-engine/.truststore \
   -storepass mypass \
   -exportcert \
   -alias cacert \
   -rfc \
   -file ca.crt
   ```

   Это создает файл сертификата с именем **ca.crt**.
3. Скопируйте сертификат на клиентскую машину с помощью команды `scp`:

```
$ scp ca.crt myuser@myclient.example.com:/home/myuser/.
```

Каждый из этих методов приводит к созданию файла сертификата с именем **ca.crt** на клиентском компьютере. Затем вы должны импортировать этот файл в хранилище сертификатов клиента.

3.2.2. Импорт сертификата в клиент

Импорт сертификата клиенту зависит от того, как клиент хранит и интерпретирует сертификаты. Дополнительную информацию об импорте сертификата см. в документации вашего клиента.

4. Общие понятия

4.1. Типы

API использует концепцию **типов** для описания различных видов принимаемых и возвращаемых объектов.

Существует три соответствующих вида типов:

- Примитивные типы - Описывают простые виды объектов, такие как string (строки) или integer (целые числа).
- Перечисляемые типы - Описывают списки допустимых значений, таких как `status` или `DiskFormat`.
- Структурированные типы - Описывают структурированные объекты с несколькими атрибутами и ссылками, например `subnet` или `network`.

4.2. Идентифицируемые типы

Многие из типов, используемых API, представляют собой идентифицируемые объекты, объекты, которые имеют уникальный идентификатор и существуют независимо от других объектов.

4.3. Объекты

Объекты — это отдельные экземпляры типов, поддерживаемых API. Например, логическая сеть с идентификатором `3d6ff755` является объектом типа **Network**.

4.4. Коллекции

Коллекция — это набор объектов одного типа.

4.5. Представления

Состояние объектов должно быть представлено при их передаче между клиентом и сервером. API поддерживает XML и JSON в качестве представления состояния объектов как для ввода, так и для вывода.

5. Сервисы

Модель REST API основана на ресурсах. Ресурс в рамках данного руководства рассматривается как сервис, выполняющий операции только над одной сущностью (entity).

Существует два соответствующих вида сервисов:

- **Сервисы, управляющие коллекцией объектов** - эти сервисы отвечают за перечисление существующих объектов и добавление новых объектов. Например, сервис **Networks** отвечает за управление набором логических сетей, доступных в системе.
- **Сервисы, управляющие конкретным объектом** - эти сервисы отвечают за получение, обновление, удаление и выполнение действий в определенных объектах. Например, сервис **Subnet** отвечает за управление конкретной логической подсетью.

Каждый сервис доступен по определенному пути внутри сервера. Например, сервис, управляющий набором логических сетей, доступных в системе, доступен по пути `/ovn/networks`, а сервис, управляющий логической сетью с идентификатором **123**, доступен по пути `/ovn/networks/123`.

Все виды сервисов имеют набор **методов**, представляющих операции, которые они могут выполнять. Сервисы, управляющие коллекциями объектов, обычно имеют методы `list` и `add`. Сервисы, управляющие определенными объектами, обычно имеют методы `get`, `update` и `remove`. Кроме того, сервисы могут также иметь методы **action**, представляющие менее распространенные операции.

Для более обычных методов существует прямое сопоставление между именем метода и именем метода HTTP:

Имя метода	HTTP-метод
`add`	POST
`get`	GET
`list`	GET
`update`	PUT

Имя метода	HTTP-метод
`remove`	DELETE

Путь, используемый в HTTP-запросе — это путь сервиса с префиксом `/zvirt-engine-backend/v3/ovn/`.

Существуют групповые сервисы, выполняющие действия над типом сущностей (например, логический маршрутизатор), такие как получить список объектов, создать новый объект, так и сервисы непосредственно конкретной сущности (сервис конкретного логического маршрутизатора), позволяющий выполнять операции над конкретной сущностью (добавить/удалить интерфейс маршрутизатора).

5.1. Особенности сервисов

Некоторые сервисы могут быть определены как самостоятельные или быть вложенными в другие сервисы. Параметры запросов к вложенным сервисам, как правило, соответствуют базовому применению.

В качестве примера рассмотрим сервис **/routers/{router:id}**, который содержит в себе вложенные сервисы:

- **/gateway** и все относящиеся к нему запросы, описанные в группе **Gateway**.

  Применение сервисов **/gateway** в сервисе **/routers/{router:id}** позволяет выполнять операции только с параметрами внешнего подключения данного логического маршрутизатора.

  Например, GET-запрос `/zvirt-engine-backend/v3/ovn/routers/{router:id}/gateway` возвращает такие параметры внешнего подключения как, описание внешней сети (network_id),параметры порта (port_id), прикрепленный IP-адрес (external_fixed_ips),информацию о шасси (gateway_chassis).

  JSON

```
{
    "network_id": "5289bbbf-7780-4b6e-9a9e-6f04bc1c084d",
    "port_id": "c14ac60d-18b0-451b-8bb6-3253c261a7a4",
    "external_fixed_ips": [
        {
            "ip_address": "10.252.22.55",
            "subnet_id": "7b9ef105-246b-4cde-b5b5-c594f277b78b",
            "link": [
                {
                    "rel": "subnet",
                    "href": "/v3/ovn/subnets/7b9ef105-246b-4cde-b5b5-
```

```
c594f277b78b"
                }
            ]
        }
    ],
    "gateway_chassis": [
        {
            "id": "6be8b4c8-9029-4795-b5f6-3706b05d7a10",
            "name": "lrp412e6583-5683-48d1-8c9c-
41e33757d79d_h1zv42.apolishchuk.local",
            "chassis_name": "h1zv42.apolishchuk.local",
            "chassis_hostname": "h1zv42.apolishchuk.local",
            "priority": 0
        }
    ],
    "mac": "02:0a:53:c7:f1:cf",
    "link": [
        {
            "rel": "network",
            "href": "/v3/ovn/external_networks/5289bbbf-7780-4b6e-9a9e-
6f04bc1c084d"
        }
    ]
}
```


- **/interfaces** и все относящиеся к нему запросы, описанные в группе **interfaces**.

  Применение сервисов **/interfaces** в сервисе **/routers/{router:id}** позволяет выполнять операции только над внутренними интерфейсами, относящимися к соответствующему логическому маршрутизатору.

Например, GET-запрос **`/zvirt-engine-backend/v3/ovn/routers/{router:id}/interfaces`** возвращает информацию о прикрепленной логической сети, подсети и IP-адресе внутреннего интерфейса маршрутизатора.

+


```
JSON | 
[
    {
        "id": "c50fd8c2-87ff-4fee-8b12-348ff91e837a",
        "network_id": "196dd3e8-c618-4ba9-a322-5a1066583359",
        "subnet_id": "c50fd8c2-87ff-4fee-8b12-348ff91e837a",
        "port_id": "6474207d-95d9-4169-8e81-e0feac52811d",
        "fixed_ip": "192.168.1.254",
        "link": [
            {
                "rel": "network",
                "href": "/v3/ovn/networks/196dd3e8-c618-4ba9-a322-5a1066583359"
            },
```

```
            {
                "rel": "subnet",
                "href": "/v3/ovn/subnets/c50fd8c2-87ff-4fee-8b12-348ff91e837a"
            }
        ]
    },
    {
        "id": "f1485029-085f-4bdd-91a2-daf238090ec0",
        "network_id": "a2fa6485-da0d-462a-91a6-debeb4312db2",
        "subnet_id": "f1485029-085f-4bdd-91a2-daf238090ec0",
        "port_id": "d6fea39c-1f70-414d-8b01-9a7a9e6469f4",
        "fixed_ip": "10.10.10.55",
        "link": [
            {
                "rel": "network",
                "href": "/v3/ovn/networks/a2fa6485-da0d-462a-91a6-debeb4312db2"
            },
            {
                "rel": "subnet",
                "href": "/v3/ovn/subnets/f1485029-085f-4bdd-91a2-daf238090ec0"
            }
        ]
    }
]
```


- **/nat** и все относящиеся к нему запросы, описанные в группе **nat**.

  Применение сервисов **/nat** в сервисе **/routers/{router:id}** позволяет выполнять операции только над механизмами NAT, относящимися к соответствующему логическому маршрутизатору.

  Например, GET-запрос **`/zvirt-engine-backend/v3/ovn/routers/{router:id}/nat`** возвращает информацию о типе NAT, IP-адреса внешнего и внутренного интерфеса используемого для преобразования для конкретного маршрутизатора.


```
JSON
[
    {
        "id": "8b33b202-e5c0-482b-8326-f484fb3c4f07",
        "external_ip": "10.252.22.23",
        "logical_ip": "192.168.1.1",
        "external_mac": "56:6f:5c:ac:00:04",
        "logical_port": "6dcf1122-4183-412f-86de-97d3d0ddb2c5",
        "type": "fip"
    }
]
```


5.2. Описание представления сервисов

Схема данных для сервисов находится в процессе исследования и описания.

Для получения недостающей информации на данный момент можно использовать выполнить GET-запрос к нужному существующему объекту и использовать результаты вывода для PUT и POST запросов.

Например, необходимо создать новую подсеть. Для этого предварительно сделайте GET-запрос к существующему списку подсетей. В теле ответа будет содержаться необходимая схема.

Пример ответа:

JSON

```
[
    {
        "id": "3c73f210-e60d-40a6-a00c-94ba896acc43",
        "name": "outsubnet2",
        "cidr": "10.252.22.0/24",
        "network_id": "377da53f-8252-4f6f-81e7-54df3e1a84d8",
        "dns_nameservers": [],
        "gateway_ip": "10.252.22.100",
        "link": [
            {
                "rel": "network",
                "href": "/v3/ovn/external_networks/377da53f-8252-4f6f-81e7-
54df3e1a84d8"
            }
        ]
    }
]
```

Из примера выше, для создания логической подсети нужны следующие данные:

- `name` - имя логической подсети (обязательно)
- `cidr` - IP-адрес подсети с маской (обязательно)
- `network_id` - идентификатор логической сети(обязательно)
- `gateway_ip` - адрес шлюза(обязательно)
- `dns_nameservers` - IP-адрес DNS сервера (не обязательно)

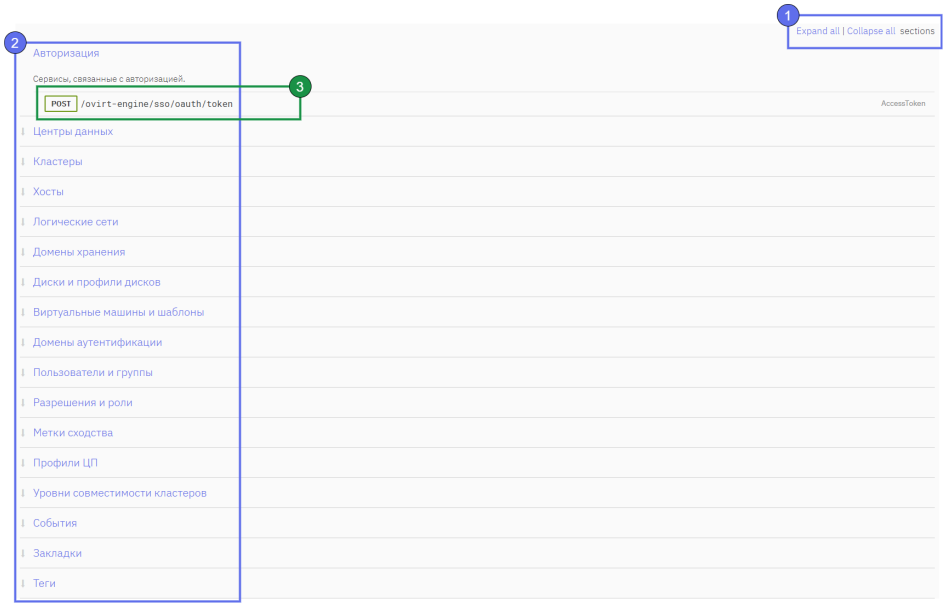
Следовательно в теле POST-запроса `/subnets` указать следующие обязательные параметры:

JSON

```
{
    "name": "API-subnet",
    "cidr": "10.252.23.0/24",
    "gateway_ip": "10.252.23.254",
    "network_id": "377da53f-8252-4f6f-81e7-54df3e1a88e2",
    "dns_nameservers": [
        "10.252.10.10"
    ]
}
```

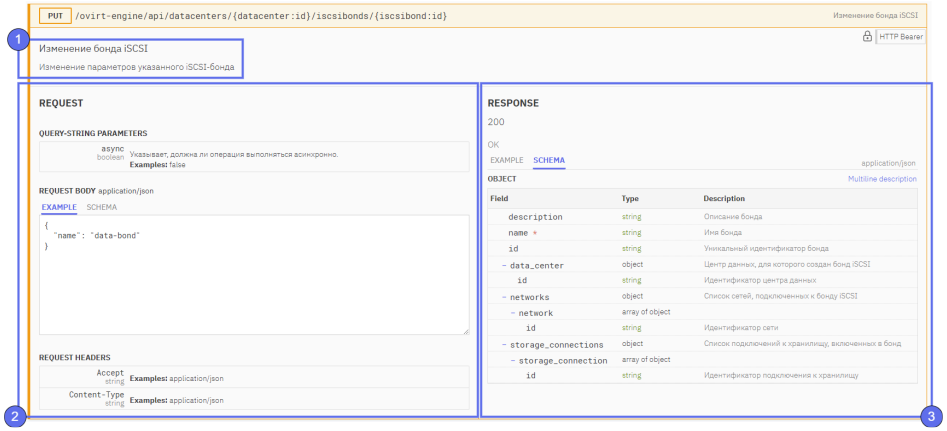
5.3. Описание интерфейса отображения сервисов

Описанные ниже методы сервисов расположены во фрейме, который содержит интерактивные элементы, обеспечивающие удобное представление.



Общие элементы управления

1. Инструменты управления отображением:
    - Expand all - позволяет развернуть все категории.
    - Collapse all - позволяет свернуть все категории.
2. Список категорий. При клике на заголовок категории можно свернуть/развернуть ее содержимое.
3. Список сервисов и их методов в категории. При клике на строку метода можно свернуть/развернуть его описание.



Элементы управления в описании методов

1. Общее и подробное описание метода.
2. Поле параметров запроса:

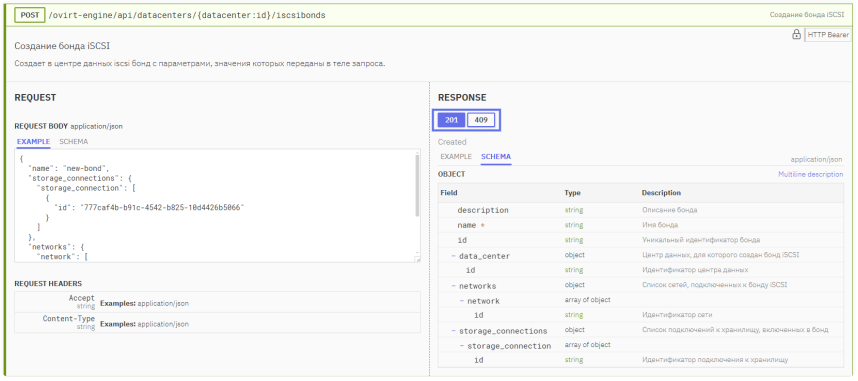
- **query-string parameters** - параметры передаваемые в строке запроса (например, `/ovirt-engine/api/datacenters?max=3`)
- **request body** - содержимое тела запроса. На вкладке **Example** представлен пример, на вкладке **Schema** - схема данных.

  Тело запроса применяется только в POST- и PUT-запросах.

  > На вкладке **Schema** представлено подробное описание параметров объекта. По умолчанию описание отображается в однострочном формате, поэтому часть описания может быть не видна. Для отображения полного формата нажмите **Multiline description**.

- **request headers** - список специфических параметров, которые необходимо включить в заголовок запроса.

3. Поле параметров ответа:
   - В верхней части содержит возвращаемые коды с описанием. В большинстве случаев представлен только пример успешного выполнения, но могут встретиться дополнительные коды (см. скрин ниже). Поскольку все методы используют одинаковый набор кодов ответа, их описание вынесено в таблицу в разделе обработка ошибок.

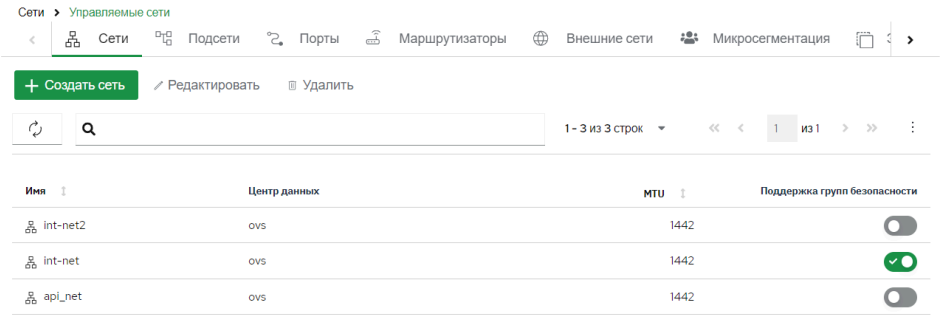


   - На вкладке **Example** представлен пример, на вкладке **Schema** - схема данных.

     > На вкладке **Schema** представлено подробное описание параметров объекта. По умолчанию описание отображается в однострочном формате, поэтому часть описания может быть не видна. Для отображения полного формата нажмите **Multiline description**.

5.4. Сервисы логических сетей

Сервис логических сетей позволяет средствами API управлять программно-определяемыми логическими сетями.



5.4.1. Описание методов логических сетей

Введите запрос

Expand all | Collapse all sections

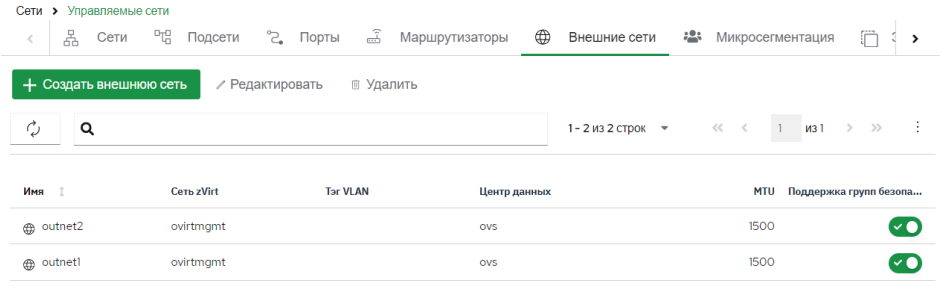
Логические сети

Сервисы, связанные с управлением логическими сетями

GET	/zvirt-engine-backend/v3/ovn/networks
POST	/zvirt-engine-backend/v3/ovn/networks
GET	/zvirt-engine-backend/v3/ovn/networks/{networks:id}
PUT	/zvirt-engine-backend/v3/ovn/networks/{networks:id}
DELETE	/zvirt-engine-backend/v3/ovn/networks/{networks:id}
GET	/zvirt-engine-backend/v3/v3/ovn/networks/csv

5.5. Сервисы внешних сетей

Описанные ниже методы, используются для управления внешними сетями средствами API.



5.5.1. Описание методов внешних сетей

Введите запрос

Expand all | Collapse all sections

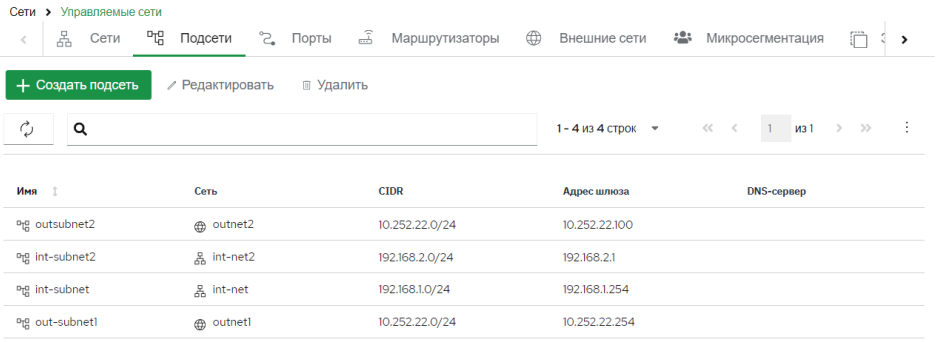
Внешние сети

Сервисы, связанные с управлением внешними сетями

GET	/zvirt-engine-backend/v3/ovn/external_networks
POST	/zvirt-engine-backend/v3/ovn/external_networks
GET	/zvirt-engine-backend/v3/ovn/external_networks/csv
GET	/zvirt-engine-backend/v3/ovn/external_networks/{id}
PUT	/zvirt-engine-backend/v3/ovn/external_networks/{id}

5.6. Сервисы подсетей

Описанные ниже методы, используются для управления подсетями средствами API.



5.6.1. Описание методов подсетей

Введите запрос

Expand all | Collapse all sections

Логические подсети

Сервисы, связанные с управление логическими подсетями

GET	/zvirt-engine-backend/v3/ovn/subnets

OVN Subnets Controller V3

POST	/zvirt-engine-backend/v3/ovn/subnets
GET	/zvirt-engine-backend/v3/ovn/subnets/csv

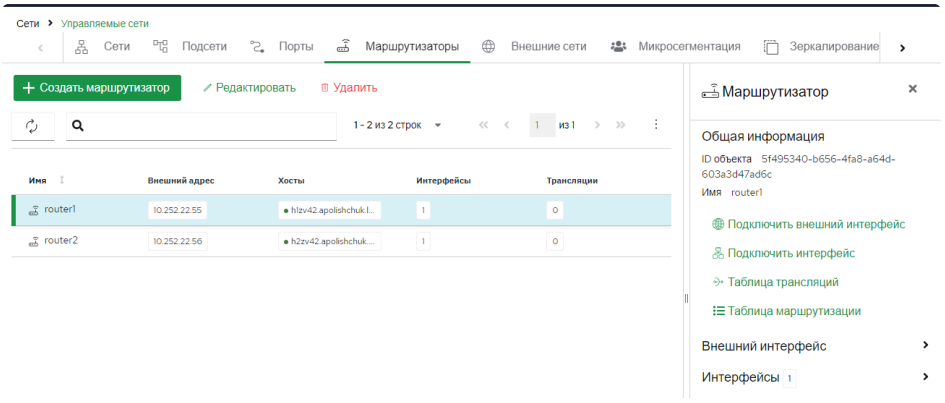
GET /zvirt-engine-backend/v3/ovn/subnets/{id}

PUT /zvirt-engine-backend/v3/ovn/subnets/{id}

DELETE /zvirt-engine-backend/v3/ovn/subnets/{id}

5.7. Сервисы маршрутизаторов

Описанные ниже методы, используются для управления логическими маршрутизаторами средствами API.



Логический маршрутизатор имеет достаточно сложную структуру корневого объекта и имеет отдельные методы по работе с вложенными сущностями, такими как механизмы NAT,таблица маршрутизации и шасси.

Свойства объекта можно поделить на следующие категории:

1. Свойства самого роутера (id, name);
2. Свойства внешнего подключения (gateway);
3. Список NAT;
4. Список маршрутов;
5. Список интерфейсов.

Вся работа с маршрутизаторами происходит непосредственно через сервис роутера, в том числе и с вложенными сущностями.

5.7.1. Описание методов маршрутизаторов

Введите запрос

Expand all | Collapse all sections

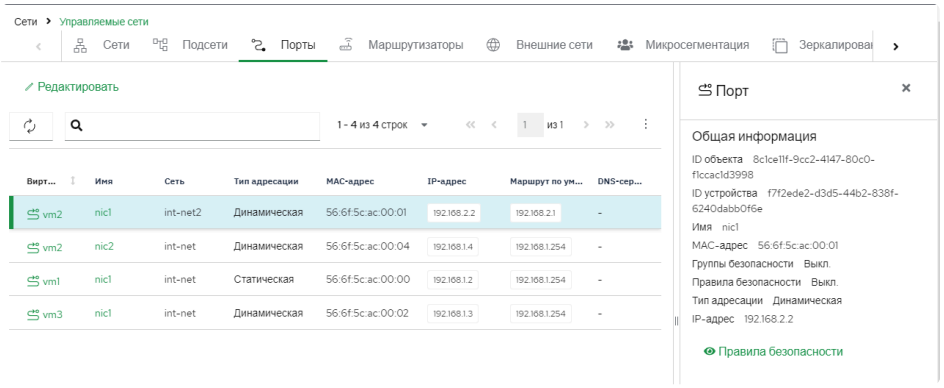
Логические маршрутизаторы

Сервисы, связанные с управлением логическим маршрутизаторами

GET	/zvirt-engine-backend/v3/ovn/routers/{id}
PUT	/zvirt-engine-backend/v3/ovn/routers/{id}
DELETE	/zvirt-engine-backend/v3/ovn/routers/{id}
GET	/zvirt-engine-backend/v3/ovn/routers/{id}/gateway
PUT	/zvirt-engine-backend/v3/ovn/routers/{id}/gateway
DELETE	/zvirt-engine-backend/v3/ovn/routers/{id}/gateway
GET	/zvirt-engine-backend/v3/ovn/routers/{id}/interfaces
POST	/zvirt-engine-backend/v3/ovn/routers/{id}/interfaces
GET	/zvirt-engine-backend/v3/ovn/routers/{id}/interfaces/{interface_id}
DELETE	/zvirt-engine-backend/v3/ovn/routers/{id}/interfaces/{interface_id}
GET	/zvirt-engine-backend/v3/ovn/routers/{id}/nat
PUT	/zvirt-engine-backend/v3/ovn/routers/{id}/nat
GET	/zvirt-engine-backend/v3/ovn/routers/{id}/routes
PUT	/zvirt-engine-backend/v3/ovn/routers/{id}/routes

5.8. Сервисы логических портов

Описанные ниже методы, используются для управления логическими портами средствами API.



Модель логического порта будет разделена на несколько дочерних объектов:

1. Список присвоенных адресов (fixed_ips);
2. Объект безопасности порта - флаг и список доступных пар MAC/IP порта;
3. Объект правил безопасности - флаг и список присвоенных групп безопасности порта;

Такое разбиение позволяет разгрузить объект порта от проверок, а также предоставить конкретные вызовы для различных видов модификации порта.

5.8.1. Описание методов логических портов

Expand all | Collapse all sections

Логические порты

Сервисы, связанные с управлением логическими портами

GET	/zvirt-engine-backend/v3/ovn/ports
GET	/zvirt-engine-backend/v3/ovn/ports/csv
GET	/zvirt-engine-backend/v3/ovn/ports/{id}
PUT	/zvirt-engine-backend/v3/ovn/ports/{id}
GET	/zvirt-engine-backend/v3/ovn/ports/{id}/assign
DELETE	/zvirt-engine-backend/v3/ovn/ports/{id}/assign
POST	/zvirt-engine-backend/v3/ovn/ports/{id}/assign/{mirrorId}
GET	/zvirt-engine-backend/v3/ovn/ports/{id}/mirrors
PUT	/zvirt-engine-backend/v3/ovn/ports/{id}/mirrors
GET	/zvirt-engine-backend/v3/ovn/ports/{id}/security
GET	/zvirt-engine-backend/v3/ovn/ports/{id}/security/acl
PUT	/zvirt-engine-backend/v3/ovn/ports/{id}/security/acl
GET	/zvirt-engine-backend/v3/ovn/ports/{id}/security/acl/groups
GET	/zvirt-engine-backend/v3/ovn/ports/{id}/security/policy
PUT	/zvirt-engine-backend/v3/ovn/ports/{id}/security/policy
GET	/zvirt-engine-backend/v3/ovn/ports/{id}/vm

5.9. Сервис правил безопасности

На данный момент в системе существует два отдельных сервиса, связанных с правилами безопасности:

1. Сервис групп безопасности;
2. Сервис правил безопасности.

Причем, правила безопасности имеют ссылку на группы безопасности, и структура сервисом повторяет аналогичную структуру в провайдере.

5.9.1. Описание методов правил безопасности

Expand all | Collapse all sections

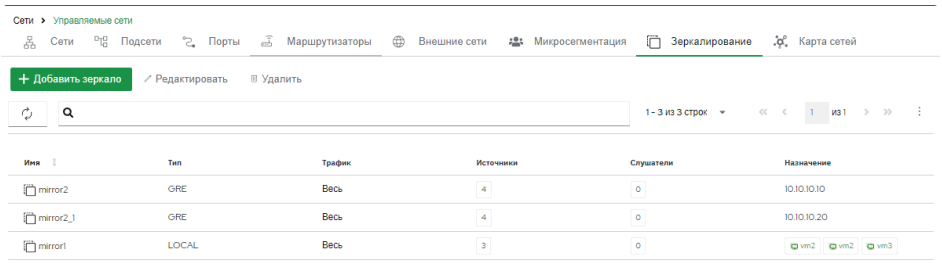
Правила и группы безопасности

Сервисы, связанные с управлением правилами и гурппами безопасности.

GET	/zvirt-engine-backend/v3/ovn/security/groups
POST	/zvirt-engine-backend/v3/ovn/security/groups
GET	/zvirt-engine-backend/v3/ovn/security/groups/csv
GET	/zvirt-engine-backend/v3/ovn/security/groups/{groupId}
PUT	/zvirt-engine-backend/v3/ovn/security/groups/{groupId}
DELETE	/zvirt-engine-backend/v3/ovn/security/groups/{groupId}
GET	/zvirt-engine-backend/v3/ovn/security/groups/{groupId}/rules
POST	/zvirt-engine-backend/v3/ovn/security/groups/{groupId}/rules
GET	/zvirt-engine-backend/v3/ovn/security/groups/{groupId}/rules/csv
GET	/zvirt-engine-backend/v3/ovn/security/groups/{groupId}/rules/{ruleId}
DELETE	/zvirt-engine-backend/v3/ovn/security/groups/{groupId}/rules/{ruleId}
GET	/zvirt-engine-backend/v3/ovn/security/rules
GET	/zvirt-engine-backend/v3/ovn/security/rules/csv
GET	/zvirt-engine-backend/v3/ovn/security/rules/{ruleId}

5.10. Сервис методов зеркалирования

Описанные ниже методы, используются для управления методами зеркалирования средствами API.



5.10.1. Описание методов зеркалирования

Введите запрос

Expand all | Collapse all sections

Зеркалирование

Сервисы, связанные с управлением механизмами зеркалирования на логических портах.

GET	/zvirt-engine-backend/v3/ovn/mirrors
POST	/zvirt-engine-backend/v3/ovn/mirrors
GET	/zvirt-engine-backend/v3/ovn/mirrors/csv
GET	/zvirt-engine-backend/v3/ovn/mirrors/{id}
PUT	/zvirt-engine-backend/v3/ovn/mirrors/{id}
DELETE	/zvirt-engine-backend/v3/ovn/mirrors/{id}
GET	/zvirt-engine-backend/v3/ovn/mirrors/{id}/listeners
GET	/zvirt-engine-backend/v3/ovn/mirrors/{id}/sources
GET	/zvirt-engine-backend/v3/ovn/ports/{id}/mirrors
PUT	/zvirt-engine-backend/v3/ovn/ports/{id}/mirrors

6. Типы данных

В этом разделе описываются примитивные типы данных, поддерживаемые API.

6.1. Тип данных string

Конечная последовательность символов Unicode.

6.2. Тип данных boolean

Представляет собой понятия "ложь (false)" и "истина (true)", используемые в математической логике.

Допустимыми значениями являются строки `false` и `true`.

Регистр игнорируется Менеджером, поэтому, например, `False` и `FALSE` также являются допустимыми значениями. Однако сервер всегда будет возвращать значения в нижнем регистре.

Для обратной совместимости со старыми версиями Менеджера также принимаются значения `0` и `1`. Значение `0` `имеет тот же смысл, что и `false`, а `1` - тот же смысл, что и `true`. Старайтесь не использовать эти значения, поскольку в будущем их поддержка может быть прекращена.

6.3. Тип данных integer

Представляет собой математическое понятие целого числа.

Допустимыми значениями являются конечные последовательности десятичных цифр.

В настоящее время Менеджер реализует этот тип с помощью знакового 32-битного целого числа, поэтому минимальное значение равно -2^{31} (-2147483648), а максимальное - $2^{31}-1$ (2147483647).

Однако в системе есть некоторые атрибуты, для которых диапазон значений, возможных при использовании 32 бит, недостаточен. В этих исключительных случаях Менеджер использует 64-битные целые числа, в частности, для следующих атрибутов:

- Disk.actual_size
- Disk.provisioned_size
- GlusterClient.bytes_read
- GlusterClient.bytes_written
- Host.max_scheduling_memory
- Host.memory
- HostNic.speed
- LogicalUnit.size

- MemoryPolicy.guaranteed
- NumaNode.memory
- QuotaStorageLimit.limit
- StorageDomain.available
- StorageDomain.used
- StorageDomain.committed
- VmBase.memory

Для этих исключений минимальное значение равно -2^{63} (-9223372036854775808), а максимальное - $2^{63}-1$ (9223372036854775807).

В будущем тип **integer** будет реализован с использованием целых чисел с неограниченной точностью, поэтому указанные выше ограничения и исключения со временем исчезнут.

6.4. Тип данных decimal

Представляет собой математическое понятие действительного числа.

В настоящее время Менеджер реализует этот тип, используя 32-битные числа IEEE 754 с плавающей запятой одинарной точности.

Для некоторых атрибутов такой точности недостаточно. В этих исключительных случаях Менеджер использует 64-битные числа с плавающей запятой двойной точности, в частности, для следующих атрибутов:

- QuotaStorageLimit.usage
- QuotaStorageLimit.memory_limit
- QuotaStorageLimit.memory_usage

В будущем десятичный тип будет реализован с использованием десятичных чисел неограниченной точности, поэтому описанные выше ограничения и исключения со временем исчезнут.

6.5. Тип данных date

Представляет собой дату и время.

Формат, возвращаемый Менеджером, соответствует формату, описанному в спецификации XML Schema при запросе XML. Например, для получения XML-представления виртуальной машины можно отправить такой запрос:

```
GET /ovirt-engine/api/vms/123
Accept: application/xml
```

Тело ответа будет содержать следующий XML-документ:

```
<vm id="123" href="/ovirt-engine/api/vms/123">
  ...
  <creation_time>2016-09-08T09:53:35.138+03:00</creation_time>
  ...
</vm>
```

При запросе JSON-представления Менеджер использует другой формат: целое число, содержащее количество миллисекунд с 1 января 1970 года, называемое также временем эпохи. Например, если послать такой запрос для получения JSON-представления виртуальной машины:

```
GET /ovirt-engine/api/vms/123
Accept: application/json
```

Тело ответа будет содержать следующий JSON-документ:

```
{
  "id": "123",
  "href="/ovirt-engine/api/vms/123",
  ...
  "creation_time": 1472564909990,
  ...
}
```

В обоих случаях даты, возвращаемые Менеджером, используют часовой пояс, настроенный на сервере, где он запущен, в приведенных примерах это UTC+3.