Управление конфигурациями

Пользователи с правами администратора могут использовать команду **openstack flavour** для настройки конфигураций и управления ими:

```
$ openstack flavor –help
```

Command "flavor" matches:

flavor create

flavor delete

flavor list

flavor set

flavor show

flavor unset

Права на конфигурацию могут быть делегированы дополнительным пользователям путём переопределения элементов управления доступом: для **os_compute_api: os-flame-manage: create, os_compute_api: os-flavour-manage: update** и **os_compute_api: os-Flame-manage: delete** в **/ etc / nova / policy .json** на сервере **nova-api.**

Настройка конфигурации может быть ограничена используемым гипервизором. Например, драйвер *libvirt* разрешает ограничения на *CPU*, доступные для виртуальной машины, настройку диска, пропускную способность ввода-вывода, поведение сторожевого таймера, управление устройством генератора случайных чисел и управление трафиком *VIF* экземпляра.

Создание конфигурации

1. Вывод списка конфигураций для просмотра их *ID* и имени, объема памяти, объема дискового пространства для корневого раздела и для временного раздела, подкачки и количества виртуальных *CPU* для каждой конфигурации:

```
$ openstack flavor list
```

2. Для создания конфигурации, укажите её имя, *ID*, размер *RAM*, размер диска, число виртуальных *CPU*:

```
$ openstack flavor create FLAVOR_NAME --id FLAVOR_ID \
--ram RAM_IN_MB --disk ROOT_DISK_IN_GB --vcpus NUMBER_OF_VCPUS
```

Уникальный *ID* (*integer* или *UUID*) для новой конфигурации: если задано "**auto**", *UUID* будет сгенерирован автоматически.

Пример создания публичной конфигурации **m1.extra_tiny** с автоматически сгенерированным *ID*, памятью размером 256 Мб, без дискового пространства, с одним *vCPU*:

```
$ openstack flavor create --public m1.extra_tiny --id auto \
--ram 256 --disk 0 --vcpus 1
```

1. Создание приватной конфигурации

Если отдельному пользователю или группе пользователей требуется особая конфигурация, к которой не должны иметь доступ другие проекты:

```
$ openstack flavor create --private m1.extra_tiny --id auto \
--ram 256 --disk 0 --vcpus 1
```

После создания конфигурации присвойте её проекту, указав имя конфигурации или *ID*, а также *ID* проекта:

```
$ openstack flavor set --project PROJECT_ID m1.extra_tiny
```

Для просмотра возможных опций используйте команду:

```
$ openstack help flavor create
```

2. Дополнительные характеристики (“*Extra specs*”)

Дополнительно для существующей конфигурации можно включить или выключить свойства, обычно называемые дополнительными характеристиками. Ключи метаданных **extra_specs** могут влиять на экземпляр виртуальной машины напрямую при его запуске. Если конфигурация устанавливает квоту **vif_outbound_peak = 65536 extra spec**, исходящая пиковая пропускная способность ввода-вывода экземпляра должна быть меньше или равна 512 Мбит/с.

Есть несколько аспектов, которые могут работать для экземпляра, включая ограничения *CPU*, настройку диска, пропускную способность ввода-вывода, поведение **watchdog** и генератор случайных чисел. Рекомендуем ознакомиться с описанием доступных ключей метаданных здесь – https://docs.openstack.org/nova/ussuri/user/flavors.html.

Для просмотра возможных опций используйте команду:

```
$ openstack flavor set --help
```

Редактирование конфигурации

Может быть отредактировано только описание конфигураций.

Чтобы отредактировать описание конфигурации, укажите её имя или *ID* и новое описание:

```
$ openstack --os-compute-api-version 2.55 flavor set --description <DESCRIPTION>
<FLAVOR>
```

Единственное обновляемое поле – это поле описания. *Nova* исторически намеренно не включала *API* для обновления конфигурации, поскольку это могло повлиять на уже созданные экземпляры виртуальных машин с этой конфигурацией. Поэтому для изменения любой другой характеристики конфигурации необходимо удалить конфигурацию и/или создать новую.

Nova хранит версию конфигурации, связанную с записью экземпляра, в таблице *instance_extra*. Хотя *Nova* поддерживает обновление *extra_specs* (updating flavor extra_specs), она не обновляет конфигурации в уже существующих экземплярах. Администраторам следует избегать обновления *extra_specs* для конфигураций, используемых существующими экземплярами. При необходимости можно использовать изменение размера для обновления существующих экземпляров, но, поскольку изменение размера выполняет «холодную» миграцию, оно не прозрачно для клиента.

Удаление конфигурации

Для удаления конфигурации укажите её имя или *ID*:

```
$ openstack flavor delete FLAVOR
```

Конфигурации по умолчанию

Предыдущие версии *Nova* имели конфигурации по умолчанию. Это было удалено из *Neuton*.

В таблице ниже перечислены конфигурации по умолчанию для релиза OpenStack *Mitaka* и более ранних версий.

Flavor	VCPUs	Disk, Гб	RAM, Мб
m1.tiny	1	1	512
m1.small	1	20	2048
m1.medium	2	40	4096
m1.large	4	80	8192
m1.xlarge	8	160	16384

Специальные метаданные при управлении конфигурациями

При создании конфигурации можно задать специальную пару ключ-значение *Extra specs*, определяющую, на какой вычислительной ноде конфигурация может примениться.

Обычно специальные метаданные *Extra specs* используются в качестве подсказок планировщика для более сложной конфигурации экземпляра. Используемые пары ключ-значение должны соответствовать известным параметрам.

Ниже перечислены опции специальных метаданных.

1. Ограничения на CPU

Позволяют задать пределы *CPU* с параметрами управления. Например, чтобы задать ограничения ввода-вывода, введите команду:

```
$ openstack flavor set FLAVOR-NAME \
--property quota:read_bytes_sec=10240000 \
--property quota:write_bytes_sec=10240000
```

Следующие необязательные параметры служат для управления долями веса, интервалами применения квот времени выполнения и квотой для максимально допустимой пропускной способности:

- **cpu_shares**

указывает пропорционально взвешенную долю для домена. Если этот элемент опущен, использует значения по умолчанию, предоставленные операционной системой. Для этого параметра нет единиц измерения, задаётся относительный параметр, основанный на настройках других виртуальных машин. Например, виртуальная машина с заданным значением **cpu_shares**, равным 2048, получает в два раза больше процессорного времени, чем виртуальная машина с заданным значением 1024.

- **cpu_shares_level**

в *VMware* параметр указывает уровень распределения. Может иметь значения `custom`, `high`, `normal`, or `low`. Если указывается `custom`, необходимо задать `cpu_shares_share`.

- **cpu_period**

задаёт интервал применения в микросекундах для гипервизоров *QEMU* и *LXC*. В течение задаваемого периода, каждый виртуальный *CPU* домена не может превысить квоту времени выполнения. Значение должно быть в диапазоне [1000, 1000000]. Период со значением 0 означает отсутствие значения.

- **cpu_limit**

задаёт верхний предел для распределения *CPU* машины *VMware* в МГц. Этот параметр гарантирует, что машина никогда не будет использовать больше, чем определенное количество процессорного времени. Его можно использовать для ограничения производительности *CPU* виртуальной машины.

- **cpu_reservation**

задаёт гарантированное минимальное резервирование *CPU* в МГц для *VMware*. Это означает, что при необходимости машине обязательно будет выделено зарезервированное количество циклов *CPU*.

- **cpu_quota**

задаёт максимально допустимую пропускную способность в микросекундах. Домен с отрицательной квотой указывает на его бесконечную пропускную способность, т. е. пропускная способность домена не контролируется. Значение должно быть в диапазоне [1000, 18446744073709551] или меньше 0. Квота со значением 0 означает отсутствие значения. Можно использовать эту функцию, для обеспечения работы всех виртуальных *CPU* с одинаковой скоростью. Например:

```
$ openstack flavor set FLAVOR-NAME \
--property quota:cpu_quota=10000 \
--property quota:cpu_period=20000
```

В этом примере экземпляр *FLAVOR-NAME* может использовать максимум 50 % ресурсов *CPU* от вычислительных возможностей физического *CPU*.

2. Ограничения на память

Предусмотрено задание ограничений на память с параметрами управления.

Следующие необязательные параметры служат для ограничения выделения памяти, гарантирования минимального резервирования памяти и указания общих ресурсов, используемых в случае конфликта ресурсов:

- **memory_limit**

задаёт верхний предел для выделения виртуальной машине памяти в мегабайтах. Использование виртуальной машиной памяти не будет превышать этот предел, даже при наличии доступных ресурсов. Обычно это используется для обеспечения стабильной производительности виртуальных машин, независимо от доступных ресурсов.

- **memory_reservation**

задаёт гарантированный минимальный объем резервирования памяти в мегабайтах для *VMware*. Это означает, что указанный объем памяти определенно будет выделен виртуальной машине.

- **memory_shares_level**

в *VMware* указывает уровень распределения. Может иметь значения **custom**, **high**, **normal** или **low**. При указании **custom**, необходимо задать **memory_shares_share**.

• **memory_shares_share**

задаёт количество общих ресурсов, выделяемых в случае использования **custom**. Единицы измерения нет, это относительный показатель, основанный на настройках других виртуальных машин. Например:

```
$ openstack flavor set FLAVOR-NAME \
--property quota:memory_shares_level=custom \
--property quota:memory_shares_share=15
```

3. Ограничения дискового ввода-вывода

Для *VMware* предусмотрена настройка ограничения ресурсов для диска с параметрами управления.

Следующие дополнительные параметры позволяют ограничить использование диска, гарантировать выделение диска и указать общие ресурсы, используемые в случае конфликта ресурсов. Это позволяет драйверу *VMware* разрешить выделение дисков для работающего экземпляра.

• **disk_io_limit**

задаёт верхний предел использования диска в количестве операций ввода-вывода в секунду. Использование виртуальной машины не будет превышать этот предел даже при наличии доступных ресурсов. Значение по умолчанию, равное -1, указывает на неограниченное использование диска.

• **disk_io_reservation**

задаёт гарантированное минимальное дисковое пространство в терминах операций ввода-вывода в секунду (*IOPS*).

• **disk_io_shares_level**

задаёт уровень распределения. Может быть выбран из значений **custom**, **high**, **normal**, or **low**. Если указывается **custom**, необходимо задать **disk_io_shares_share**.

• **disk_io_shares_share**

задаёт количество общих ресурсов, выделяемых в случае использования **custom**. При конкуренции за ресурсы значение используется для определения распределения ресурсов.

Пример, в котором для **disk_io_reservation** устанавливается значение 2000 *IOPS*:

```
$ openstack flavor set FLAVOR-NAME \
--property quota:disk_io_reservation=2000
```

4. Настройка диска

Используя квоты дискового ввода-вывода, можно задать максимальную скорость записи на диск 10 Мб/с для пользователя виртуальной машины. Например:

```
$ openstack flavor set FLAVOR-NAME \
--property quota:disk_write_bytes_sec=10485760
```

Опции дискового ввода-вывода:

disk_read_bytes_sec

disk_read_iops_sec

disk_write_bytes_sec

disk_write_iops_sec

disk_total_bytes_sec

disk_total_iops_sec

5. Пропускная способность ввода-вывода

Опции ввода-вывода виртуального интерфейса (vif):

- **vif_inbound_average**
- **vif_inbound_burst**
- **vif_inbound_peak**
- **vif_outbound_average**
- **vif_outbound_burst**
- **vif_outbound_peak**

Входящий и исходящий трафики могут формироваться независимо. Элемент полосы пропускания может иметь не более одного входящего и не более одного исходящего дочернего элемента. Если не включить какой-либо из этих дочерних элементов, качество обслуживания (*QoS*) не будет применяться к этому направлению трафика. Для формирования только входящего трафика сети следует использовать только входящий дочерний элемент (и наоборот). Каждый элемент имеет один обязательный атрибут **average**, определяющий среднюю скорость передачи данных на формируемом интерфейсе.

Также есть два необязательных целочисленных атрибута: **peak**, задающий максимальную скорость, с которой мост может отправлять данные (в килобайтах / секунду), и **burst** – количество байтов, которые могут быть отправлены при максимальной скорости (килобайты). Скорость распределяется поровну внутри доменов, подключенных к сети.

Пример задания ограничения пропускной способности сетевого трафика для существующей конфигурации представлен ниже.

Исходящий трафик:

average: 262 Мбит/с (32768 килобайт/с)

peak: 524 Мбит/с (65536 килобайт/с)

burst: 65536 килобайт

Входящий трафик:

average: 262 Мбит/с (32768 килобайт/с)

peak: 524 Мбит/с (65536 килобайт/с)

burst: 65536 килобайт

```
$ openstack flavor set FLAVOR-NAME \
--property quota:vif_outbound_average=32768 \
--property quota:vif_outbound_peak=65536 \
--property quota:vif_outbound_burst=65536 \
--property quota:vif_inbound_average=32768 \
--property quota:vif_inbound_peak=65536 \
--property quota:vif_inbound_burst=65536
```

Все значения ограничения скорости в приведенном примере указаны в килобайтах в секунду. И значения пакета указаны в килобайтах. Значения были преобразованы с использованием Data rate units on Wikipedia.

6. Аппаратная видеопамять

Управление максимальным размером памяти (RAM) видеоизображения задается параметром **hw_video: ram_max_mb** вместе со свойством изображения **hw_video_ram**, который должен быть меньше или равен **hw_video: ram_max_mb**.

В настоящее время поддержка осуществляется драйверами **libvirt** и **vmware**.

Дополнительная информация об установке атрибута **vram** с драйвером **libvirt** – https://libvirt.org/formatdomain.html#elementsvideo.

Дополнительная информация об установке атрибута **videoRamSizeInKB** с драйвером **vmware** – https://pubs.vmware.com/vi-sdk/visdk250/ReferenceGuide/vim.vm.device.VirtualVideoCard.html

7. Поведение сторожевого таймера

Для драйвера **libvirt** предусмотрена возможность включения и задания поведения устройства сторожевого таймера виртуального оборудования для каждой конфигурации. Сторожевые устройства следят за гостевым сервером и выполняют настроенное действие при зависании сервера. Сторожевой таймер использует устройство i6300esb (имитирующее PCI Intel 6300ESB). Сторожевой таймер отключен, если **hw: watchdog_action** не задан.

Задание поведения сторожевого таймера осуществляется вводом команды:

```
$ openstack flavor set FLAVOR-NAME --property hw:watchdog_action=ACTION
```

Допустимые значения **ACTION**:

- `disabled (по умолчанию)` – устройство не подключено.
- `reset` – принудительный перезапуск гостевого сервера.
- `poweroff` – принудительное отключение гостевого сервера.
- `pause` – приостановка гостевого сервера.
- `none` – включение только сторожевого таймера; отсутствие действий при зависании сервера.

Примечание – Поведение сторожевого таймера, заданное с помощью свойств определенного изображения, переопределит поведение, заданное с помощью конфигураций.

8. Генератор случайных чисел

Если устройство генератора случайных чисел добавлено к экземпляру через его свойства изображения, устройство можно включить и настроить с помощью ввода кманды:

```
$ openstack flavor set FLAVOR-NAME \
--property hw_rng:allowed=True \
--property hw_rng:rate_bytes=RATE-BYTES \
--property hw_rng:rate_period=RATE-PERIOD
```

где **RATE-BYTES** (целое число) – разрешенное количество байтов, которое гость может прочитать из энтропии хоста за период;

RATE-PERIOD (целое число) – продолжительность периода чтения в миллисекундах.

9. Блок мониторинга производительности (vPMU)

Если *nova* развернута с драйвером *libvirt virt* и для **libvirt.virt_type** задано значение **qemu** или **kvm**, vPMU можно включить или отключить для экземпляра с помощью **hw: pmu extra_spec** или свойства изображения **hw_pmu**. Поддерживаемые значения: **True** или **False**. Если *vPMU* не

включен или не отключен явно через разновидность или образ, решение о его наличии остается на усмотрение *QEMU*.

```
$ openstack flavor set FLAVOR-NAME --property hw:pmu=True|False
```

Блок мониторинга производительности используется такими инструментами, как *perf* в гостевой системе, для предоставления более точной информации для профилирования приложения и мониторинга производительности гостевой системы. Для рабочих нагрузок в реальном времени эмуляция *vPMU* может вызвать дополнительную задержку, которая может быть нежелательной. Если предоставляемая телеметрия не требуется, для таких рабочих нагрузок следует установить **`hw: pmu = False`**. Для большинства рабочих нагрузок правильным будет значение по умолчанию не задано или включено *vPMU* **`hw: pmu = True`**.

10. Топология CPU

Для драйвера *libvirt* можно задать топологию процессоров на виртуальной машине с помощью свойств. Свойства с указанием *max* ограничивают количество, выбираемое пользователем с помощью свойств изображения.

```
$ openstack flavor set FLAVOR-NAME \
--property hw:cpu_sockets=FLAVOR-SOCKETS \
--property hw:cpu_cores=FLAVOR-CORES \
--property hw:cpu_threads=FLAVOR-THREADS \
--property hw:cpu_max_sockets=FLAVOR-SOCKETS \
--property hw:cpu_max_cores=FLAVOR-CORES \
--property hw:cpu_max_threads=FLAVOR-THREADS
```

где **`FLAVOR-SOCKETS`** (целое число) – количество сокетов гостевой виртуальной машины. По умолчанию установлено значение количества запрошенных виртуальных CPU.

`FLAVOR-CORES` (целое число) – количество ядер на сокет для гостевой виртуальной машины. По умолчанию установлено значение 1.

`FLAVOR-THREADS`:(целое число) – количество потоков на ядро для гостевой виртуальной машины. По умолчанию установлено значение 1.

11. Политика закрепления CPU

Для драйвера *libvirt* можно привязать виртуальные *CPU* экземпляров к физическим ядрам *CPU* хоста с помощью свойств. Можно дополнительно задать, как используются аппаратные потоки *CPU* в архитектуре на основе одновременной многопоточности (*SMT*). Эти конфигурации приведут к улучшенному детерминированию и производительности для каждого экземпляра.

Архитектуры на основе *SMT* включают процессоры *Intel* с технологией *Hyper-Threading*. В этих архитектурах ядра процессора совместно используют ряд компонентов с одним или несколькими другими ядрами. Ядра в таких архитектурах обычно называются аппаратными потоками, в то время как ядра, с которыми данное ядро совместно используют компоненты, называются родственными потоками.

Для отделения закрепленных экземпляров от незакрепленных следует использовать агрегаты узлов, поскольку последние не будут соответствовать требованиям к ресурсам первого.

```
$ openstack flavor set FLAVOR-NAME \
--property hw:cpu_policy=CPU-POLICY \
--property hw:cpu_thread_policy=CPU-THREAD-POLICY
```

Допустимые значения **CPU-POLICY**:

shared: (по умолчанию) – гостевым *vCPU* будет разрешено свободно перемещаться по хосту *pCPU*, хотя это может быть ограничено политикой *NUMA*.

dedicated: гостевые виртуальные *CPU* будут строго привязаны к набору узловых виртуальных *CPU*. При отсутствии явного запроса топологии *vCPU* драйверы обычно предоставляют все *vCPU* как сокеты с одним ядром и одним потоком. Когда действует строгое закрепление *CPU*, топология гостевого *CPU* будет настроена в соответствии с топологией *CPU*, к которым он привязан. Этот вариант подразумевает коэффициент превышения 1.0. Например, если "гость" с двумя виртуальными *CPU* привязан к одному ядру хоста с двумя потоками, тогда "гость" получит топологию из одного сокета, одного ядра и двух потоков.

Допустимые значения **CPU-THREAD-POLICY**:

prefer (по умолчанию) – хост может иметь или не иметь архитектуру *SMT*. Для SMT-архитектуры предпочтительны родственные потоки.

isolate – хост не должен иметь архитектуру *SMT* или должен имитировать архитектуру, отличную от *SMT*. Если хост не имеет архитектуры *SMT*, каждый виртуальный *CPU* размещается на другом ядре, как и ожидалось. У хоста с архитектурой *SMT* одно или несколько ядер имеют родственные потоки, то каждый виртуальный *CPU* размещается на другом физическом ядре. Никакие виртуальные *CPU* от других "гостей" не размещаются на одном ядре. Таким образом, гарантируется невозможность использования всех потоков, кроме одного, на каждом используемом ядре.

require – хост должен иметь архитектуру *SMT*. Каждый виртуальный *CPU* распределяется по родственным потокам. Хост без архитектуры *SMT* не используется. Для хоста с архитектурой *SMT*, но недостаточным числом доступных ядер со свободными родственными потоками, планирование не выполняется.

Параметр **hw: cpu_thread_policy** действителен, только если для **hw: cpu_policy** установлено значение **dedicated**.

12. Политика PCI NUMA Affinity

Для драйвера *libvirt* можно указать политику *NUMA affinity* для сквозных устройств *PCI* и интерфейсов *neutron SR-IOV* через дополнительные метаданные конфигурации **hw: pci_numa_affinity_policy** или свойство образа ***hw_pci_numa_affinity_policy***.

Допустимые значения: **required**,``**preferred**`` или **legacy** (по умолчанию):

required – *Nova* загружает экземпляры с устройствами *PCI*, только если хотя бы один из узлов *NUMA* экземпляра связан с этими устройствами *PCI*. Если информация об узле *NUMA* для некоторых устройств *PCI* не может быть определена, эти устройства *PCI* не будут использоваться экземпляром. Таким образом обеспечивается максимальная производительность.

preferred – *Nova-scheduler* выберет вычислительный хост с минимальным учетом *NUMA* affinity устройств *PCI*. *Nova-compute* попытается наилучшим образом выбрать устройства *PCI* на основе *NUMA affinity*, однако, если это невозможно, *nova-compute* вернется к планированию на узле *NUMA*, который не связан с устройством *PCI*.

legacy (по умолчанию) – описывает текущее поведение *Nova*. Обычно имеется информация об ассоциации устройств *PCI* с узлами *NUMA*. Однако некоторые устройства *PCI* не предоставляют такую информацию. Значение *legacy* означает, что *Nova* будет загружать экземпляры с устройства *PCI*, если:

- устройство *PCI* связано как минимум с одним узлом *NUMA*, на котором будет загружаться экземпляр.
- отсутствует информации о доступных *PCI* с *NUMA affinity*.

13. Топология NUMA

Для драйвера *libvirt* можно определить размещение *NUMA*-хоста для потоков виртуальных *CPU* экземпляра, а также выделение виртуальных *CPU* экземпляра и памяти из *NUMA*-узлов хоста. Для конфигураций, размер памяти и выделение *vCPU* которых больше размера *NUMA*-узлов в вычислительных хостах, определение топологии *NUMA* позволяет хостам эффективнее использовать *NUMA* и повысить производительность операционной системы экземпляра.

```
$ openstack flavor set FLAVOR-NAME \
--property hw:numa_nodes=FLAVOR-NODES \
--property hw:numa_cpus.N=FLAVOR-CORES \
```

--property hw:numa_mem.N=FLAVOR-MEMORY

где **FLAVOR-NODES** (целое число) – количество *NUMA*-узлов для ограничения выполнения потоков *vCPU* экземпляра. Если не задано, потоки *vCPU* могут выполняться на любом количестве доступных NUMA-узлов.

N (целое число): *NUMA*-узел экземпляра для применения заданного *CPU* или памяти, где *N* находится в диапазоне от 0 до (*FLAVOR-NODES* - 1).

FLAVOR-CORES (список целых чисел, разделенных запятыми) – список виртуальных *CPU* экземпляра для сопоставления с *NUMA*-узлом *N*-экземпляра. Если не указан, виртуальные *CPU* равномерно распределяются между доступными *NUMA*-узлами.

FLAVOR-MEMORY (целое число) – количество мегабайт памяти экземпляра для сопоставления с *NUMA*-узлом *N*-экземпляра. Если не указан, виртуальные *CPU* равномерно распределяются между доступными *NUMA*-узлами.

hw: numa_cpus.N и **hw: numa_mem.N** действительны, только если установлен параметр **hw: numa_nodes**. Кроме того, они требуются только в том случае, если *NUMA*-узлы экземпляра имеют асимметричное распределение процессоров и оперативной памяти (что важно для некоторых рабочих нагрузок *NFV*).

Параметр *N* является индексом гостевых *NUMA*-узлов и может не соответствовать *NUMA*-узлам хоста. Например, на платформе с двумя *NUMA*-узлами планировщик может выбрать размещение гостевого *NUMA*-узла 0, как указано в **hw: numa_mem.0**, на *NUMA*-узле 1 хоста, и наоборот. Точно так же целые числа, используемые для *FLAVOR-CORES*, являются индексами гостевых виртуальных *CPU* и могут не соответствовать центральным *CPU*. Таким образом, эту функцию нельзя использовать для ограничения экземпляров конкретными центральными процессорами или *NUMA*-узлами.

Если объединенные значения **hw: numa_cpus.N** или **hw: numa_mem.N** больше доступного количества процессоров или памяти соответственно, возникает исключение.

14. Аппаратное шифрование памяти "гостя"

При наличии вычислительных узлов, поддерживающих шифрование памяти "гостя" на аппаратном уровне, эту функциональность можно вызвать с помощью дополнительного параметра спецификации **hw: mem_encryption**:

```
$ openstack flavor set FLAVOR-NAME \
--property hw:mem_encryption=True
```

15. Политика CPU реального времени

Для драйвера *libvirt* можно указать, что один или несколько виртуальных *CPU* экземпляра, но не все они, работают с политикой реального времени. При использовании на правильно настроенном хосте это реализует более надежное обеспечение наихудшего случая задержки планировщика для *vCPU* и является требованием для определенных приложений.

```
$ openstack flavor set FLAVOR-NAME \
--property hw:cpu_realtime=CPU-REALTIME-POLICY \
--property hw:cpu_realtime_mask=CPU-REALTIME-MASK
```

где

CPU-REALTIME-POLICY – параметр, имеющий одно из следующих значений:

no (по умолчанию) – гостевые виртуальные *CPU* не имеют политики реального времени;

yes – гостевые виртуальные *CPU* имеют политики реального времени;

CPU-REALTIME-MASK (основная маска) – маска ядра, указывающая, какие виртуальные CPU не будут иметь политики реального времени. Значение должно начинаться с символа ^. Например, значение ^ 0-1 указывает, что все виртуальные *CPU*, кроме виртуальных *CPU* 0 и 1, имеют политику реального времени.

Опция **hw:cpu_realtime_mask** действительна, только если для параметра **hw:cpu_realtime** установлено значение **yes**.

Рекомендуется документировать необходимые шаги для настройки хостов и "гостей". Необходимо многое: от изоляции хостов и конфигурирования опций конфигурации *Nova* [*compute*] **cpu_dedicated_set** на хосте до выбора правильно настроенного гостевого образа.

Важно! Хотя большинство виртуальных *CPU* экземпляра могут работать с политикой реального времени, хотя бы один виртуальный *CPU* должен быть помечен как не работающий в режиме реального времени, чтобы использовать его как для гостевых процессов, не работающих в режиме реального времени, так и для служебных процессов эмулятора.

Важно! Для использования этой опции должны быть включены закрепленные процессоры (см. раздел «Политика закрепления CPU»).

16. Политика потоков эмулятора

Для драйвера *libvirt* можно назначить отдельный *pCPU* экземпляру, который будет использоваться для потоков эмулятора, то есть процессов эмулятора, не связанных напрямую с гостевой операционной системой. Этот *pCPU* будет использован в дополнение к *pCPU*,

задействованного для гостя. Обычно это требуется для применения с рабочей нагрузкой в реальном времени.

Важно! При применении этих специальных метаданных, должны быть включены закреплённые *CPU* (см. раздел «Политика закрепления CPU»).

```
$ openstack flavor set FLAVOR-NAME \
--property hw:emulator_threads_policy=THREAD-POLICY
```

Ожидаемое поведение потоков эмулятора зависит от значения метаданных конфигурации **hw:emulator_threads_policy** и значения **compute.cpu_shared_set**. Они представлены в таблице ниже.

cpu_shared_set (стринговая) – маска центральных процессоров, которые могут использоваться для ресурсов *vCPU* и выгруженных потоков эмулятора.

Возможные значения – список перечисленных через запятую значений, где каждое значение – либо отдельный номер *CPU*, либо интервал номеров, либо символ вставки, за которым следует номер *CPU*, который должен быть исключен из предыдущего диапазона. Например:

```
cpu_shared_set = "4-12, ^ 8,15
```

	compute.cpu_shared_set	**compute.cpu_shared_unset**
hw:emulator_treads_policy unset (по умолчанию)	Прикреплено ко всем *pCPU* экземпляра	Прикреплено ко всем *pCPU* экземпляра
hw:emulator_threads_policy = share	Прикреплено к **compute.cpu_shared_set**	Прикреплено ко всем *pCPU* экземпляра
hw:emulator_threads_policy = isolate	Прикреплено к одному *pCPU*, отличному от *pCPU* экземпляра	Прикреплено к одному *pCPU*, отличному от *pCPU* экземпляра