РЕДСОФТ

Операционная система «РЕД ОС»

Руководство администратора

Часть 3

RU.29926343.02.01-01 32 1-3

Copyright © 2025 ООО "РЕД СОФТ"

HTTPS://REDOS.RED-SOFT.RU/

Настоящий документ является третьей частью руководства администратора операционной системы «РЕД ОС» версии 8 (далее РЕД ОС, ОС) для процессоров архитектур `x86_64`. В данной части руководства администратора рассматриваются и описываются действия по администрированию средств виртуализации (`libvirt`) и контейнеризации (`docker`, `kubernetes`), а также ключевые возможности сканера уязвимостей `trivy`.

В данной части руководства администратора рассмотрены следующие ключевые моменты:

- процесс установки и настройки средств виртуализации и контейнеризации;
- формат используемых в настройке файлов конфигурации;
- основные команды, необходимые для управления средствами виртуализации и контейнеризации и их компонентами;
- основные компоненты обеспечения безопасности средств виртуализации и контейнеризации;
- основные принципы журналирования событий средств виртуализации и контейнеризации;
- а также ключевые возможности сканера уязвимостей `trivy`.

Руководство администратора часть 3 предназначено для системных администраторов или инженеров, имеющих опыт эксплуатации средств виртуализации и контейнеризации.

Октябрь 2025

Оглавление

Средство виртуализации

1 Средство виртуализации . . . 7

1.1 Общие сведения . . . 7

1.2 Средства управления средой виртуализации . . . 7

1.2.1 Утилита qemu-img . . . 8

1.2.2 Утилита virt-install . . . 10

1.2.3 Утилита virsh . . . 11

1.2.4 Менеджер виртуальных машин virt-manager . . . 14

1.3 Установка среды виртуализации . . . 14

1.3.1 Установка в процессе развертывания новой системы РЕД ОС . . . 15

1.3.2 Установка в существующей системе РЕД ОС . . . 17

1.4 Ролевой доступ . . . 17

1.4.1 Создание новых ролей . . . 18

1.4.2 Объекты и разрешения . . . 18

1.4.3 Создание пользователей . . . 21

1.4.4 Назначение и отзыв ролей . . . 22

1.5 Подключение к гипервизору . . . 23

1.5.1 Подключение к гипервизору через SSH . . . 23

1.5.2 Подключение к гипервизору с помощью virsh . . . 23

1.5.3 Подключение к гипервизору с помощью менеджера ВМ . . . 25

1.6 Создание виртуальных машин . . . 26

1.6.1 Создание ВМ с помощью утилиты virt-install . . . 26

1.6.2 Создание ВМ с помощью менеджера virt-manager . . . 27

1.6.3 Создание ВМ через файл конфигурации . . . 32

1.7 Управление ВМ . . . 36

1.7.1 Управление ВМ с помощью утилиты virsh . . . 36

1.7.2 Управление ВМ с помощью менеджера virt-manager . . . 44

1.8 Безопасность в среде виртуализации . . . 58

1.8.1 Доверенная загрузка ВМ . . . 58

1.8.2 Контроль целостности ВМ . . . 59

1.8.3 Ограничение программной среды . . . 59

1.8.4 Резервное копирование . . . 59

1.8.5 Защита памяти . . . 61

1.8.6 Управление потоками информации . . . 62
1.8.7 Регистрация событий безопасности . . . 66

Средства контейнеризации

2 Средство контейнеризации . . . 70
2.1 Общие сведения . . . 70
2.2 Установка среды контейнеризации . . . 71
2.2.1 Установка в процессе развертывания новой системы РЕД ОС . . . 71
2.2.2 Установка в существующей системе РЕД ОС . . . 73
2.3 Доступ к среде контейнеризации . . . 73
2.4 Компоненты средства контейнеризации . . . 73
2.4.1 Dockerfile . . . 74
2.4.2 Docker Registry . . . 79
2.4.3 Docker Daemon . . . 79
2.4.4 Docker Client . . . 80
2.4.5 Объекты Docker . . . 80
2.4.6 Docker volume . . . 81
2.5 Управление контейнерами . . . 84
2.5.1 Docker Container . . . 84
2.5.2 Docker build . . . 86
2.5.3 Docker image . . . 86
2.5.4 Docker images . . . 87
2.5.5 Docker create . . . 87
2.5.6 Docker run . . . 89
2.5.7 Docker start . . . 90
2.5.8 Docker stop . . . 90
2.5.9 Docker pause . . . 91
2.5.10 Docker login . . . 91
2.5.11 Docker attach . . . 92
2.5.12 Docker exec . . . 92
2.5.13 Docker logs . . . 93
2.5.14 Docker stats . . . 94
2.5.15 Docker tag . . . 95
2.6 Безопасность в среде контейнеризации . . . 95
2.6.1 Изоляция контейнеров . . . 95
2.6.2 Выявление уязвимостей в образах контейнеров . . . 97
2.6.3 Проверка корректности конфигурации контейнеров . . . 98
2.6.4 Контроль целостности контейнеров и их образов . . . 100
2.6.5 Регистрация событий безопасности . . . 102

3 Платформа Kubernetes . . . 105
3.1 Основные понятия . . . 105
3.2 Доступ к среде контейнеризации . . . 105

3.3 Компоненты Kubernetes . . . 106
3.3.1 Компоненты управляющего узла . . . 106
3.3.2 Компоненты узла . . . 107
3.3.3 Образы контейнеров для создания минимального кластера Kubernetes в закрытом контуре . . . 108
3.4 Управление кластерами . . . 108
3.4.1 Управление кластерами с помощью kubeadm . . . 108
3.4.2 Манифесты . . . 110
3.4.3 Управление кластером с помощью kubectl . . . 113
3.5 Предварительные условия . . . 117
3.5.1 Требования к среде развертывания . . . 117
3.5.2 Входные данные . . . 117
3.5.3 Минимально необходимые полномочия контейнеров . . . 118
3.6 Развертывание Docker Registry . . . 120
3.6.1 Настройка сети . . . 120
3.6.2 Установка docker . . . 120
3.6.3 Настройка и запуск локального registry . . . 120
3.6.4 Загрузка образов . . . 122
3.7 Настройка Master-node . . . 123
3.8 Настройка рабочих нод . . . 136
3.9 Подключение рабочей ноды к кластеру . . . 139
3.10 Безопасность в среде контейнеризации Kubernetes . . . 140
3.10.1 Изоляция контейнеров . . . 140
3.10.2 Выявление уязвимостей в образах контейнеров . . . 140
3.10.3 Проверка корректности конфигурации контейнеров . . . 140
3.10.4 Контроль целостности контейнеров и их образов . . . 144
3.10.5 Регистрация событий безопасности . . . 146

4 Сканер уязвимостей Trivy . . . 149
4.1 Общие сведения . . . 149
4.2 Установка . . . 149
4.3 Сканирование локальной файловой системы . . . 151
4.4 Сканирование образов контейнеров . . . 153
4.4.1 Сканирование файлов внутри образа контейнера . . . 153
4.4.2 Сканирование метаданных образа контейнера . . . 155
4.5 Сканирование кластера Kubernetes . . . 156
4.6 Очистка кеша утилиты . . . 159
4.7 Регистрация событий безопасности . . . 159

Средство виртуализации

1. Средство виртуализации

1.1 Общие сведения

Виртуализация — предоставление набора вычислительных ресурсов или их логического объединения, независимое от аппаратной реализации и обеспечивающее при этом логическую изоляцию друг от друга вычислительных процессов, выполняемых на одном физическом ресурсе.

Поддержка виртуализации в РЕД ОС обеспечивается технологией KVM (Kernel-based Virtual Machine). Программное обеспечение KVM состоит из загружаемого модуля ядра `kvm.ko`, предоставляющего базовый сервис виртуализации, процессорно-специфического загружаемого модуля `kvm-amd.ko` или `kvm-intel.ko` и компонентов пользовательского режима QEMU.

QEMU — средство эмуляции аппаратного обеспечения различных платформ, поддерживающее также аппаратную виртуализацию, основанную на технологии Intel-VT или AMD-V, на процессорах Intel и AMD.

Управление средой виртуализации в РЕД ОС осуществляется посредством набора инструментов `libvirt`, предоставляющего единый API к различным технологиям виртуализации.

1.2 Средства управления средой виртуализации

Управление средой виртуализации осуществляется с помощью сервера виртуализации `libvirt`. Главным компонентом сервера виртуализации является служба `libvirtd`.

Для выполнения необходимых задач управления гостевыми системами служба `libvirtd` должна быть запущена на хостовых ОС (в данном случае РЕД ОС Сервер). Под задачами управления понимаются такие действия, как создание, запуск, остановка, мониторинг и контроль ВМ, а также настройка сети, управление хранилищем и образами дисков.

Основным файлом конфигурации сервера виртуализации является файл `/etc/libvirt/libvirtd.conf`. В нем задаются настройки и параметры, необходимые для корректной работы службы `libvirtd`, такие как настройка безопасных сетевых соединений, политика разграничения доступа, способы аутентификации, уровень журналирования и пр.

Далее приведено описание ключевых параметров файла конфигурации `/etc/libvirt/libvirtd.conf`:

- `listen_tls = 0` — флаг прослушивания безопасных `TLS`-соединений с использованием сертификатов. По умолчанию параметр включен, чтобы отключить —

раскоммертируйте строку;
- `listen_tcp = 1` — флаг прослушивания незашифрованных TCP-соединений. По умолчанию параметр отключен, чтобы включить — раскомментируйте строку;
- `tls_port = "16514", tcp_port = "16509"` — номера портов для обеспечения сетевого соединения;
- `listen_addr = "192.168.0.1"` — IP-адрес или имя хоста сетевого интерфейса для приема соединений;
- `auth_tcp = "sasl", auth_tls = "none"` — способ аутентификации для сокетов TCP и TLS. Могут принимать значения:
  - `none` — не выполнять проверку;
  - `sasl` — использовать структуру `sasl`;
  - `polkit` — использовать политику `polkit`.
- `log_level = 3` — уровень журналирования. Может принимать значения:
  - `4` — ошибка;
  - `3` — предупреждение;
  - `2` — информационное сообщение;
  - `1` — отладочное сообщение.

Помимо основного файла конфигурации сервер виртуализации взаимодействует со следующими каталогами файловой системы хостовой ОС:
- `/var/lib/libvirt` — рабочий каталог сервера виртуализации;
- `/var/log/libvirt` — каталог журналов сервера виртуализации.

Для управления средой виртуализации могут использоваться как консольные, так и графические утилиты.

Набор инструментов `libvirt` включает в себя следующие основные компоненты управления:
- `virt-install` — утилита командной строки, предназначенная для создания ВМ;
- `virt-manager` — графическая утилита, предназначенная для управления ВМ;
- `virsh` — утилита командной строки, предназначенная для управления ВМ, сетями и дисками.

Для управления образами ВМ используется утилита `qemu-img`, входящая в состав средства эмуляции аппаратного обеспечения QEMU.

1.2.1 Утилита qemu-img

Утилита `qemu-img` входит в состав пакета `qemu-kvm` и предназначена для манипуляций с образами дисков ВМ, таких как форматирование, создание и преобразование.

`Qemu-img` поддерживает работу со следующими форматами образов дисков:
- `raw` — используется по умолчанию, достоинствами являются простота и возможность экспортирования в другие эмуляторы;
- `qcow2` — формат QEMU, наиболее гибкий формат, рекомендуется использовать для небольших образов;
- `qcow` — старый формат QEMU, используется только в целях обеспечения совместимости со старыми версиями;
- `cow` — формат COW (Copy On Write), используется только в целях обеспечения совместимости со старыми версиями;
- `vmdk` — формат образов, совместимый с VMware 3 и 4;

- `cloop` — формат `CLOOP` (Compressed Loop);
- `vpc` (`vhd`) — формат, поддерживающий полную структуру и содержание, сходные с жёстким диском.

Для манипуляций с образами дисков используются следующие команды:
- `create` — создание нового образа диска;
- `check` — проверка образа диска на ошибки;
- `convert` — конвертация существующего образа диска в другой формат;
- `info` — получение информации о существующем образе диска;
- `snapshot` — управление снимками состояний (snapshot) существующих образов дисков;
- `commit` — запись произведенных изменений на существующий образ диска;
- `rebase` — создание нового базового образа на основании существующего.

Синтаксис утилиты имеет следующий вид:

```
qemu-img <команда> [<опции>]
```

С подробной информацией об использовании утилиты можно ознакомиться, выполнив команду:

```
man qemu-img
```

Примеры использования утилиты:
1. Получение информации об образе:

```
qemu-img info /var/lib/libvirt/images/ro8.qcow2
```

```
image: /var/lib/libvirt/images/ro8.qcow2
file format: qcow2
virtual size: 5 GiB (5368709120 bytes)
disk size: 5 GiB
cluster_size: 65536
Format specific information:
    compat: 1.1
    compression type: zlib
    lazy refcounts: true
    refcount bits: 16
    corrupt: false
    extended l2: false
```

2. Создание нового образа:

```
qemu-img create -f qcow2 /var/lib/libvirt/images/redos_vm.qcow2 20G
```

3. Преобразование формата существующего образа:

```
qemu-img convert -f vpc vpc_image.vhd -O raw raw_image.dsk
```

1.2.2 Утилита virt-install

Утилита `virt-install` является инструментом командной строки и позволяет создавать ВМ с использованием библиотеки управления гипервизором `libvirt`. Конфигурационные файлы ВМ сохраняются в виде `XML`-файла. После создания ВМ сохраненный конфигурационный `XML`-файл можно отредактировать.

Утилита `virt-install` поддерживает графическую установку с использованием `VNC` или `SPICE`, а также установку в текстовом режиме из консоли. Гостевая ОС может быть настроена на использование одного или нескольких виртуальных дисков, сетевых интерфейсов, аудиоустройств, а также физических `USB`- или `PCI`-устройств.

Установочный носитель может храниться как локально (`ISO`-образ или `CD-ROM`), так и на удаленном сервере (`HTTP`, `HTTPS` или `FTP`). В случае расположения установочного носителя на `FTP`-сервере утилита получит минимальный набор файлов для запуска процесса установки, позволяя получать пакеты по мере необходимости. Кроме того поддерживается загрузка компонентов гостевых ОС по сети (`PXE`) и импорт образа диска с возможностью создания ВМ без запуска процесса установки.

С помощью соответствующих опций утилиты `virt-install` можно автоматизировать процесс создания виртуальных машин.

Большинство опций не являются обязательными для заполнения. Если при создании ВМ указано значение для параметра `--os-variant`, остальные параметры будут заполнены автоматически. Если параметр `--os-variant` не указан, минимально необходимыми опциями для создания ВМ являются:

- `--name` — имя новой ВМ;
- `--ram` — размер ОЗУ;
- `--disk` или `--nodisks` — настройка хранилища.

Установка утилиты осуществляется командой:

```
dnf install virt-install
```

Синтаксис утилиты имеет следующий вид:

```
virt-install [<опции>]
```

Далее будут рассмотрены ключевые опции утилиты `virt-install`.

Подключение к серверу виртуализации:

- `--connect <URI>` — подключение к гипервизору не по умолчанию.
  Допустимые варианты:
  - `qemu:///system` — для создания ВМ KVM и QEMU, которые будут запускаться системным экземпляром `libvirtd`. Данный режим является режимом по умолчанию;
  - `qemu:///session` — для создания ВМ KVM и QEMU для `libvirtd`, запущенной от имени обычного пользователя.

Общие параметры:

- `-n <имя_ВМ>`, `--name=<имя_ВМ>` — имя новой ВМ, должно быть уникальным в рамках одного гипервизора;
- `--memory <опции>` — память, выделяемая для ВМ (в МБ);

- `--vcpus=<число>` — количество виртуальных процессоров для ВМ. Могут быть указаны некоторые дополнительные параметры:
  - `maxvcpus=MAX` — максимальное кол-во виртуальных процессоров, выделяемых для ВМ;
  - `sockets=#` — количество сокетов;
  - `cores=#` — количество ядер;
  - `threads=#` — количество потоков выполнения.
- `--cpu <параметры>` — модель ЦП и его параметры, например: `--cpu host` или `--cpu core2duo,+x2apic,disable=vmx`.

Параметры установки:
- `-c`, `--cdrom <путь>` — путь к `ISO`-файлу или устройство `CD-ROM` для использования в качестве установочного носителя ВМ;
- `-l`, `--location <путь>` — источник установки дистрибутива, например, `https://host/path`;
- `--pxe` — установка из сети с использованием `PXE`;
- `--import` — пропустить процесс установки ОС и создать гостевую машину на основе существующего образа диска;
- `--disk <параметры>` — настройка параметров хранилища ВМ, таких как тип носителя, размер (в ГБ), формат диска и пр.;
- `--filesystem <каталог>` — экспорт каталога хостовой ОС в ВМ;
- `-w <параметры>`, `--network <параметры>` — настройка сети для ВМ;
- `--vnc` — позволяет открыть графическое окно установки виртуальной машины.

С подробной информацией об использовании утилиты можно ознакомиться, выполнив команду:

```
man virt-install
```

1.2.3 Утилита virsh

Утилита `virsh` — инструмент командной строки, предназначенный для управления ВМ и гипервизором KVM.

Утилита `virsh` использует API `libvirt` и является альтернативой графического менеджера ВМ `virt-manager`.

С помощью `virsh` можно управлять жизненным циклом ВМ, хранилищами данных, виртуальными сетями, сохранять состояние ВМ и переносить ВМ между гипервизорами.

Синтаксис утилиты имеет следующий вид:

```
virsh [<параметры>] ... [<команда_в_интерактивном_режиме>]
virsh [<параметры>] ... <команда> [<агрументы>]
```

Используемые параметры:
- `-c`, `--connect=<URI>` — `URI` подключения к гипервизору;
- `-d`, `--debug=<ЧИСЛО>` — уровень отладки [0-4];
- `-e`, `--escape <символ>` — управляющая последовательность для консольной команды;
- `-h`, `--help` — справка;

- -k, --keepalive-interval=<СЕК> — интервал отправки сообщений keepalive в секундах, 0 — отключает механизм keepalive;
- -K, --keepalive-count=<ЧИСЛО> — максимально допустимое число пропущенных сообщений keepalive;
- -l, --log=<ФАЙЛ> — вывод сообщений в файл;
- -q, --quiet — режим без уведомлений;
- -r, --readonly — подключиться в режиме чтения;
- -t, --timing — показывать время команд;
- -v — краткая версия;
- -V — подробная информация о версии;
- --version[=<ТИП>] — <ТИП> может принимать значения short (краткий номер версии) и long (подробная информация). По умолчанию используется значение short.

Команды управления ВМ:
- list — просмотр всех виртуальных машин;
- guestinfo — запросить информацию о гостевой системе (с помощью агента);
- create — создать виртуальную машину из файла конфигурации XML и запустить ее;
- start — запустить неактивную виртуальную машину;
- shutdown — корректно завершить работу виртуальной машины;
- destroy — принудительно остановить работу виртуальной машины;
- reboot — перезагрузить виртуальную машину;
- suspend — приостановить работу виртуальной машины;
- resume — возобновить работу приостановленной виртуальной машины;
- define — определить файл конфигурации XML для заданной виртуальной машины;
- dumpxml — вывести файл конфигурации XML для заданной виртуальной машины;
- domid — просмотр идентификатора виртуальной машины;
- domuuid — просмотр UUID виртуальной машины;
- dominfo — просмотр сведений о виртуальной машине;
- domname — просмотр имени виртуальной машины;
- domstate — просмотр состояния виртуальной машины;
- restore — восстановить сохраненную в файле виртуальную машину;
- save — сохранить состояние виртуальной машины в файл;
- undefine — удалить все файлы виртуальной машины.

Команды управления ресурсами ВМ:
- attach-device — подключить определенное в XML-файле устройство к ВМ;
- attach-disk — подключить новое дисковое устройство к ВМ;
- attach-interface — подключить новый сетевой интерфейс к ВМ;
- detach-device — отключить устройство от ВМ (принимает те же определения XML, что и attach-device);
- detach-disk — отключить дисковое устройство от ВМ;
- detach-interface — отключить сетевой интерфейс от ВМ;
- setmem — определить размер выделенной ВМ памяти;
- setmaxmem — ограничить максимально доступный гипервизору объем памяти;
- setvcpus — изменить число предоставленных ВМ виртуальных процессоров;

- `vcpuinfo` — просмотр информации о виртуальных процессорах;
- `vcpupin` — настройка соответствий виртуальных процессоров;
- `domblkstat` — получить статистику блочного устройства домена;
- `domifstat` — получить статистику сетевого интерфейса домена;
- `domblklist` — список блочных устройств домена;
- `dominfo` — информация о домене;
- `snapshot-create` — создать снимок на основе `XML`;
- `snapshot-create-as` — создать снимок на основе набора аргументов;
- `snapshot-delete` — удалить снимок домена;
- `snapshot-list` — показать снимки домена;
- `snapshot-revert` — восстановить домен из снимка;
- `pool-create-as` — создать пул на основе набора аргументов;
- `pool-create` — создать пул из файла `XML`;
- `pool-list` — список пулов;
- `vol-create-as` — создать том на основе набора аргументов;
- `vol-create` — создать том из файла `XML`;
- `vol-delete` — удалить том;
- `vol-list` — список томов;
- `vol-resize` — изменить размер тома;
- `vol-download` — загрузить содержимое тома в файл;
- `vol-upload` — отправить содержимое файла в том;
- `vol-wipe` — очистить том.

Ознакомиться со списком всех доступных команд или параметров можно, используя команду:

```
virsh --help
```

Примеры использования утилиты:
1. Подключение к локальному гипервизору:

```
virsh --connect qemu:///system
```

2. Сохранение `XML`-файла конфигурации виртуальной машины:

```
virsh dumpxml redos-8
```

Команда выводит содержимое `XML`-файла в стандартный вывод (`stdout`), для сохранения данных можно перенаправить вывод в файл, например:

```
virsh dumpxml redos-8 > ro8.xml
```

3. Создание виртуальной машины на основе файла конфигурации:

```
virsh create ro8.xml
```

4. Отправка сигнала на завершение работы виртуальной машины:

```
virsh shutdown redos-8
```

1.2.4 Менеджер виртуальных машин virt-manager

Менеджер виртуальных машин `virt-manager` предоставляет графический интерфейс для доступа к гипервизорам и виртуальным машинам как в локальной, так и в удаленных системах.

Графический менеджер `virt-manager` позволяет выполнять следующие задачи управления ВМ:

- создание, редактирование, запуск и остановка виртуальных машин;
- просмотр и управление каждой виртуальной машиной посредством консоли;
- просмотр производительности и статистики по каждой виртуальной машине;
- просмотр всех запущенных виртуальных машин на хосте и их производительность;
- использование виртуальных машин, запущенных локально или удаленно.

Установка менеджера ВМ производится командой:

```
dnf install virt-manager
```

После установки инструмент будет доступен из «Главного меню» — «Системные» — «Менеджер виртуальных машин».

Главное окно менеджера выглядит следующим образом:

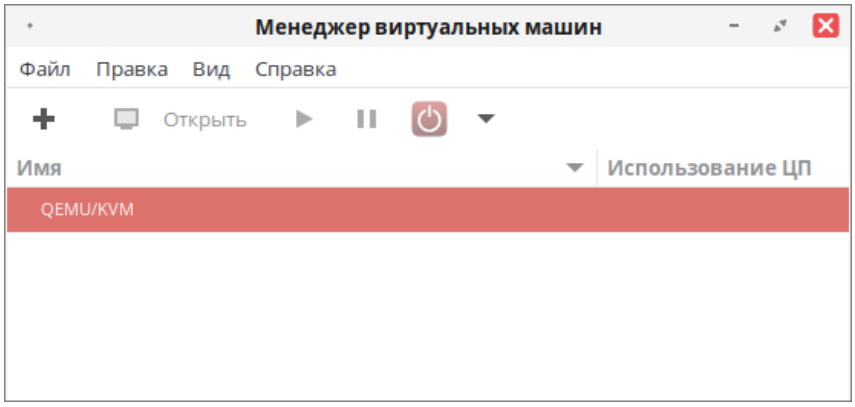

Рисунок 1.1 — Окно Менеджера виртуальных машин

Здесь отображаются доступные подключения и установленные ВМ, а также используемые ими ресурсы. Для открытия графической консоли и параметров ВМ необходимо дважды кликнуть по ее имени.

В основном меню менеджера можно настроить подключение к гипервизору, создать новые ВМ, просмотреть параметры существующих ВМ и ознакомиться с информацией о программе.

На панели инструментов доступно создание ВМ и управление их состоянием (запуск, пауза, выключение).

1.3 Установка среды виртуализации

Существует два варианта установки среды виртуализации:

- установка в процессе развертывания новой системы РЕД ОС;
- установка в существующей системе РЕД ОС.

1.3.1 Установка в процессе развертывания новой системы РЕД ОС

После запуска интерактивной установки РЕД ОС с установочного носителя (`USB`, `CD-ROM`, `PXE`) параметры будущей системы настраиваются в штатном режиме (выбор языка, часового пояса, раскладки клавиатуры, источника установки, настройка сети), кроме параметра «Выбор программ».

Для развертывания среды виртуализации необходимо выполнить установку РЕД ОС конфигурации Сервер (с графическим интерфейсом или минимальный).

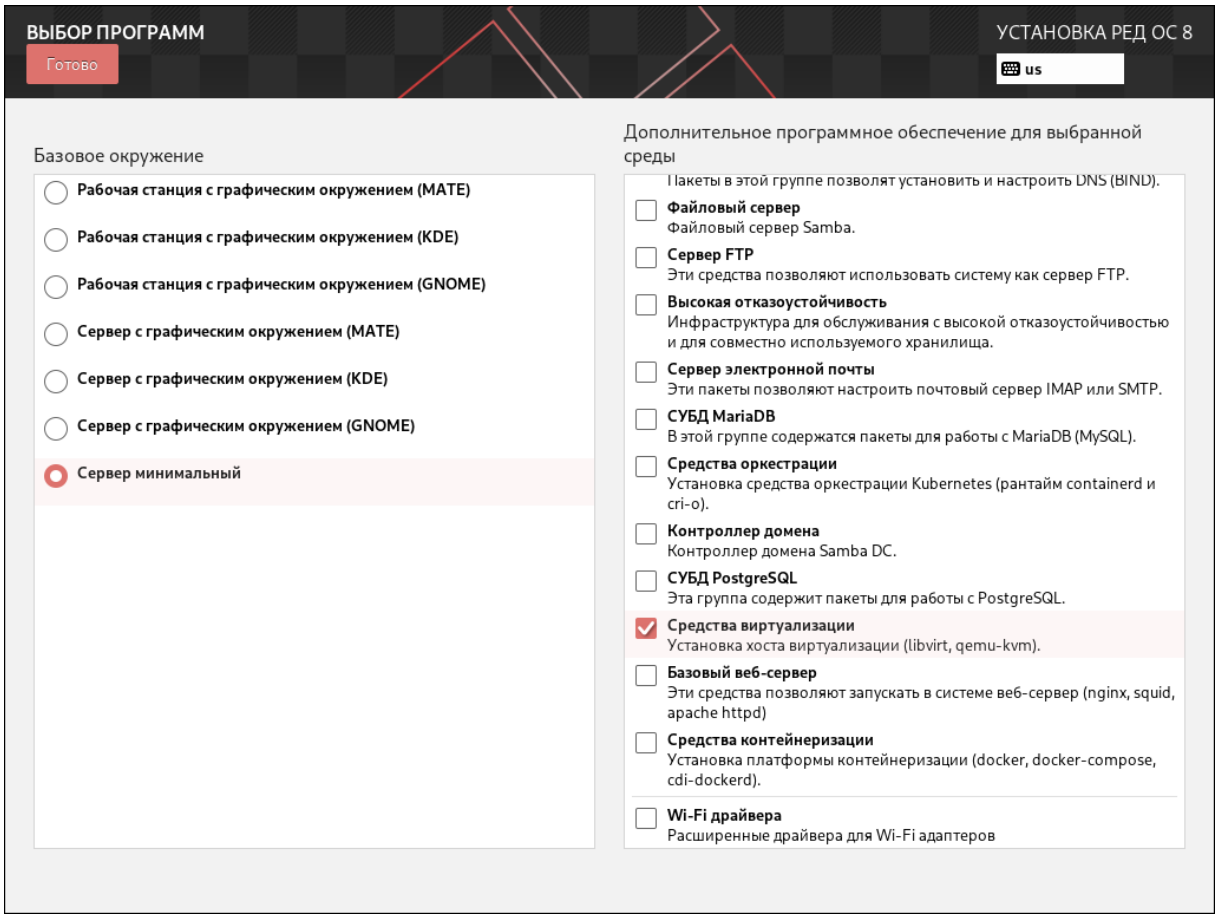

Рисунок 1.2 — Установка средства виртуализации

При выборе конфигурации Сервер (с графическим интерфейсом или минимальный) следует определить дополнительное программное обеспечение — *Средства виртуализации*.

Далее необходимо задать локального пользователя системы и его пароль, а также настроить пароль для суперпользователя `root`. Для возможности удаленного подключения к серверу виртуализации необходимо установить флаг в поле «Разрешить вход пользователем root с паролем через SSH».

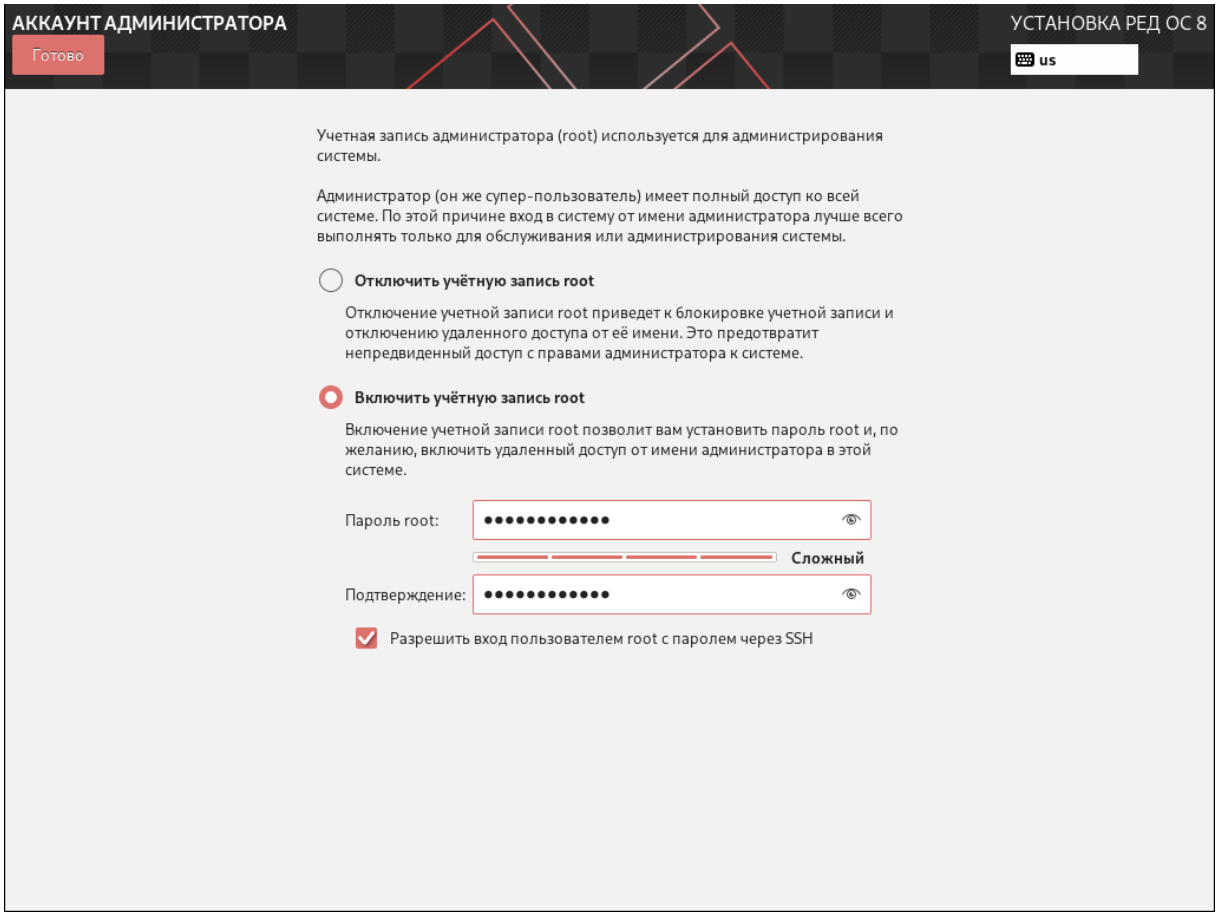

Рисунок 1.3 — Создание пароля суперпользователя root

После определения всех настроек нажмите кнопку «Начать установку».

Для завершения процесса установки потребуется перезагрузить систему и принять Лицензионное соглашение.

После авторизации в системе необходимо проверить статус службы `libvirtd` командой:

```
systemctl status libvirtd
```

В статусе должно быть указано `active (running)`. Если по каким-либо причинам статус не активен, выполните команду запуска службы с правами пользователя `root`:

```
systemctl start libvirtd
```

После этого ОС будет готова к использованию.

1.3.2 Установка в существующей системе РЕД ОС

Сначала необходимо проверить, поддерживает ли ваш процессор аппаратную виртуализацию. Проверка осуществляется командой:

```
egrep '(vmx|svm)' /proc/cpuinfo
```

Если вывод команды не пуст, значит процессор поддерживает аппаратную виртуализацию.

После проверки можно приступать к развертыванию среды виртуализации.

Для установки пакетов сервера виртуализации `libvirt` выполните команду:

```
dnf install libvirt qemu-kvm virt-install virt-manager \
libvirt-cert-profile
```

Затем запустите и добавьте в автозагрузку службу сервера `libvirtd`:

```
systemctl enable libvirtd --now
```

Проверьте статус службы сервера `libvirtd`:

```
systemctl status libvirtd
```

В статусе должно быть указано `active (running)`.

1.4 Ролевой доступ

В средстве виртуализации реализован ролевой метод управления доступом. Используется 4 основные роли пользователей:

- разработчик виртуальной машины — `vm-developer`;
- администратор безопасности средства виртуализации — `security-admin`;
- администратор средства виртуализации — `superuser`;
- администратор виртуальных машин — `vm-admin`.

Каждая роль подразумевает ряд предопределенных прав доступа и управления средой виртуализации.

Роль разработчика виртуальной машины (`vm-developer`) позволяет:

- создавать виртуальные машины;
- изменять конфигурации виртуальных машин.

Роль администратора безопасности средства виртуализации (`security-admin`) позволяет:

- иметь доступ на чтение к журналу событий безопасности средства виртуализации;
- формировать отчеты с учетом заданных критериев отбора, выгрузку (экспорт) данных из журнала событий безопасности средства виртуализации.

Роль администратора средства виртуализации (`superuser`) позволяет:

- создавать учетные записи пользователей средства виртуализации;
- управлять учетными записями пользователей средства виртуализации;

- назначать права доступа пользователям средства виртуализации к виртуальным машинам;
- создавать и удалять виртуальное оборудование средства виртуализации;
- изменять конфигурации виртуального оборудования средства виртуализации;
- управлять доступом виртуальных машин к физическому и виртуальному оборудованию;
- управлять квотами доступа виртуальных машин к физическому и виртуальному оборудованию;
- управлять перемещением виртуальных машин;
- удалять виртуальные машины;
- запускать и останавливать виртуальные машины;
- создавать снимки состояния виртуальных машин, включающих файл конфигурации виртуальной машины, образа виртуальной машины и образа памяти виртуальной машины.

Роль администратора виртуальной машины (`vm-admin`) позволяет:

- осуществлять доступ пользователя средства виртуализации к виртуальной машине посредством интерфейса средства виртуализации.

1.4.1 Создание новых ролей

Средство виртуализации также обеспечивает возможность определения полномочий для пользователей средства виртуализации в пределах назначенных им ролей.

Файл, определяющий полномочия пользователей, должен быть создан в каталоге `/etc/libvirt/privileges.d/` и иметь название `roles.user`. Доступ к файлу может иметь только суперпользователь `root`.

Создание новых ролей производится вручную посредством внесения изменений в файл `/etc/libvirt/privileges.d/roles.user`. Для создания новой роли необходимо добавить наименование роли и правила разрешений в формате `org.libvirt.api.<объект>.<разрешение>`.

Создание новой роли осуществляется по следующему шаблону:

```
nano /etc/libvirt/privileges.d/roles.user
```

```
"<имя_роли>": [
    "org.libvirt.api.<объект>.<разрешение>",
    "org.libvirt.api.<объект>.<разрешение>",
    "org.libvirt.api.<объект>.<разрешение>",
    ...    ...    ...
    "org.libvirt.api.<объект>.<разрешение>"
  ]
```

1.4.2 Объекты и разрешения

Средство виртуализации применяет контроль доступа ко всем основным типам объектов в своем API. Каждый тип объекта, в свою очередь, имеет определенный набор разрешений. Далее будут рассмотрены основные объекты и разрешения средства виртуализации.

Объект connect

Объект `connect` определяет настройки соединения. Для него существуют следующие разрешения:

- `detect-storage-pools` — определение пулов хранения;
- `getattr` — доступ к соединению;
- `interface-transaction` — транзакции интерфейса;
- `pm-control` — управление питанием хоста;
- `read` — чтение конфигурации соединения;
- `search-domains` — список доменов;
- `search-interfaces` — список интерфейсов;
- `search-networks` — список сетей;
- `search-node-devices` — список устройств хоста;
- `search-nwfilter-bindings` — список привязок сетевых фильтров;
- `search-nwfilters` — список сетевых фильтров;
- `search-secrets` — список секретов;
- `search-storage-pools` — список пулов хранения;
- `write` — запись конфигурации соединения на хост.

Объект domain

Объект `domain` определяет настройки доступа к домену (ВМ). Для него существуют следующие разрешения:

- `block-read` — чтение из блочных устройств домена;
- `block-write` — запись в блочные устройства домена;
- `checkpoint` — создание контрольной точки;
- `core-dump` — создание `coredump`-файла при аварийном завершении домена;
- `delete` — удаление домена;
- `fs-freeze` — заморозка файловой системы домена;
- `fs-trim` — очистка удаленных блоков на блочном устройстве домена
- `getattr` — доступ к домену;
- `hibernate` — ввод домена в спящий режим;
- `init-control` — разрешение, необходимое для управления выключением/перезагрузкой гостевой ОС;
- `inject-nmi` — отправка `nonmasked-interrupt` домену;
- `mem-read` — чтение памяти домена;
- `migrate` — миграция домена;
- `open-device` — разрешение на открытие последовательного канала/графической консоли домена;
- `open-graphics` — разрешение открытия графической консоли;
- `pm-control` — управление питанием;
- `read` — чтение конфигурации домена;
- `read-secure` — разрешение на чтение конфигурации домена, с включением секретной информации (например, пароли для доступа к блочным устройствам);
- `reset` — перезагрузка домена;
- `save` — сохранение состояния домена;
- `screenshot` — создание скриншота домена;
- `send-input` — разрешение на отправку кодов клавиш;
- `send-signal` — разрешение на отправку сигналов определённым процессам внутри домена;

- `set-password` — установка паролей пользователей внутри домена;
- `set-time` — установка времени в домене;
- `snapshot` — создание снапшотов домена;
- `start` — запуск домена;
- `stop` — остановка домена;
- `suspend` — приостановка домена;
- `write` — запись конфигурации домена.

Объект interface

Объект `interface` определяет настройки доступа к интерфейсам. Для него существуют следующие разрешения:

- `delete` — удаление интерфейса;
- `getattr` — доступ к интерфейсу;
- `read` — чтение конфигурации интерфейса;
- `save` — сохранение интерфейса;
- `start` — запуск интерфейса;
- `stop` — остановка интерфейса;
- `write` — запись конфигурации интерфейса.

Объект network

Объект `network` определяет настройки доступа к сети. Для него существуют следующие разрешения:

- `delete` — удаление настроек сети;
- `getattr` — доступ к настройкам сети;
- `read` — чтение настроек сети;
- `save` — сохранение настроек сети;
- `search-ports` — список доступных сетевых портов;
- `start` — запуск сети;
- `stop` — остановка сети;
- `write` — запись сетевых настроек.

Объект network-port

Объект `network-port` определяет настройки доступа к сетевым портам. Для него существуют следующие разрешения:

- `create` — назначение сетевых портов;
- `delete` — удаление сетевых портов;
- `read` — чтение конфигурации сетевых портов;
- `write` — запись конфигурации сетевых портов.

Объект node-device

Объект `node-device` определяет настройки доступа к устройствам хоста. Для него существуют следующие разрешения:

- `delete` — удаление устройства хоста;
- `detach` — отсоединение устройства хоста;
- `getattr` — доступ к устройствам хоста;
- `read` — чтение конфигурации устройств хоста;
- `start` — запуск устройств хоста;
- `stop` — остановка устройства хоста;
- `write` — запись конфигурации устройства хоста.

Объект nwfilter

Объект `nwfilter` определяет настройки доступа к фильтрам сети. Для него существуют следующие разрешения:

- `delete` — удалить фильтр сети;
- `getattr` — доступ к настройке фильтра сети;
- `read` — чтение настроек фильтра сети;
- `save` — сохранение настроек фильтра сети;
- `write` — запись настроек фильтра сети.

Объект storage-pool

Объект `storage-pool` определяет настройки доступа к пулу хранения. Для него существуют следующие разрешения:

- `delete` — удаление пула хранения;
- `format` — формат пула хранения;
- `getattr` — доступ к пулу хранения;
- `read` — чтение конфигурации пула хранения;
- `refresh` — обновление пула хранения;
- `save` — сохранение пула хранения;
- `search-storage-vols` — список томов в пуле хранения;
- `start` — запуск пула хранения;
- `stop` — остановка пула хранения;
- `write` — запись конфигурации пула хранения.

Объект storage-vol

Объект `storage-vol` определяет настройки доступа к томам хранения. Для него существуют следующие разрешения:

- `create` — создание пула хранения;
- `data-read` — чтение данных из тома хранения;
- `data-write` — запись данных в тома хранения;
- `delete` — удаление тома хранения;
- `format` — формат тома хранения;
- `getattr` — доступ к томам хранения;
- `read` — чтение конфигурации тома хранения;
- `write` — запись конфигурации тома;
- `resize` — изменение размера тома хранения.

1.4.3 Создание пользователей

В среде виртуализации изначально должен быть создан хотя бы один пользователь — администратор средства виртуализации (`superuser`). Для его создания необходимо с правами пользователя `root` выполнить команду:

```
useradd <имя_пользователя>
```

и задать созданному пользователю пароль:

```
passwd <имя_пользователя>
```

Далее необходимо назначить данному пользователю роль администратора средства виртуализации (`superuser`) командой:

```
libvirt-rbac-tool assign-role <имя_пользователя> superuser
```

Все остальные пользователи создаются администратором средства виртуализации (`superuser`) с помощью утилиты `libvirt-dac-tool`.

Синтаксис утилиты:

```
libvirt-dac-tool <команда> <имя_пользователя>
```

Доступные команды утилиты:

- `useradd` — создание нового пользователя;
- `passwd` — создание пароля пользователю;
- `usermod` — добавление пользователя в группу;
- `userdel` — удаление пользователя.

Приведенные выше команды по структуре аналогичны одноименным системным командам. Для просмотра подробных сведений о командах и их доступных параметрах выполните:

```
man <команда>
или
<команда> --help
```

Например:

```
man useradd
```

1.4.4 Назначение и отзыв ролей

Для настройки прав доступа пользователей к средству виртуализации предназначена утилита `libvirt-rbac-tool`. Права доступа пользователям может назначать только администратор средства виртуализации (`superuser`).

Для назначения какой-либо роли пользователю средства виртуализации выполните команду:

```
libvirt-rbac-tool assign-role <имя_пользователя> <имя_роли>
```

Для отзыва какой-либо роли у пользователя выполните команду:

```
libvirt-rbac-tool revoke-role <имя_пользователя> <имя_роли>
```

Для просмотра назначенных пользователю ролей выполните команду:

```
libvirt-rbac-tool list-user-roles <имя_пользователя>
```

Для того чтобы назначить пользователю права доступа к какой-либо ВМ, выполните команду:

```
libvirt-rbac-tool assign-role-to-vm <имя_пользователя> <имя_роли>
<имя_ВМ>
```

Для отзыва у пользователя прав доступа к какой-либо ВМ выполните команду:

```
libvirt-rbac-tool revoke-role-from-vm <имя_пользователя> <имя_
роли> <имя_ВМ>
```

Для просмотра назначенных прав доступа к ВМ выполните команду:

```
libvirt-rbac-tool list-vm-roles <имя_ВМ>
```

1.5 Подключение к гипервизору

Подключение к гипервизору виртуализации может осуществляться тремя способами:

1. С использованием безопасного протокола удаленного доступа SSH.
2. С помощью утилиты командной строки virsh.
3. Через графический менеджер ВМ virt-manager.

1.5.1 Подключение к гипервизору через SSH

Для подключения к гипервизору виртуальных машин с использованием протокола безопасных соединений SSH на клиентском ПК, с которого планируется подключение, необходимо сгенерировать SSH-ключ. Публичную часть ключа следует скопировать на сервер виртуализации.

Для этого на клиентском ПК необходимо выполнить следующие команды с правами того пользователя, от имени которого будет осуществляться подключение:

```
ssh-keygen -t rsa
ssh-copy-id <имя_пользователя>@<IP-адрес_сервера_виртуализации>
```

Для доступа к среде виртуализации подключающемуся пользователю потребуется назначить соответствующие права. Подробную информацию о настройке прав доступа см. в разделе 1.4 «Ролевой доступ» настоящего руководства.

Также для доступа к среде виртуализации можно использовать учетную запись пользователя root. Для этого необходимо скопировать публичный ключ для root и выполнить подключение к серверу:

```
ssh-copy-id root@<IP-адрес_сервера_виртуализации>
ssh root@<IP-адрес_сервера_виртуализации>
```

1.5.2 Подключение к гипервизору с помощью virsh

Подключение к гипервизору можно осуществить с помощью утилиты командной строки virsh.

Команда подключения выглядит следующим образом:

```
virsh --connect <URI>
```

где параметр <URI> может принимать следующие значения:

- qemu:///system – для локального подключения к службе, управляющей доменами KVM/QEMU. Данный вариант используется по умолчанию;

- `qemu:///session` – для локального подключения к службе, управляющей доменами KVM/QEMU и запущенной от имени непривилегированного пользователя;
- `qemu+ssh://<имя_пользователя>@<IP-адрес_сервера>/system` — для подключения к удаленной службе, управляющей доменами KVM/QEMU.

После успешного подключения будет открыт интерактивный терминал гипервизора, в котором непосредственно выполняются команды управления ВМ и их ресурсами.

Для удобства обращения можно настроить переменные окружения, благодаря которым не будет необходимости каждый раз заново подключаться к гипервизору и открывать его интерактивный терминал, вся работа будет выполняться сразу из терминала пользователя средства виртуализации.

Для настройки переменных окружения с правами пользователя `root` создайте файл `/etc/environment`:

```
nano /etc/environment
```

и добавьте в него следующую строку:

```
LIBVIRT_DEFAULT_URI=<URI>
```

где `<URI>` может принимать значения, описанные выше.

Настройку можно произвести для подключения как к локальному гипервизору, так и к удаленному.

Также есть возможность установить соединение только для чтения, для этого необходимо в команде указать параметр `--readonly`.

Примеры использования:

1. Создание локального подключения к среде виртуализации:

```
virsh --connect qemu:///system list --all
```

```
 ID   Имя            Состояние
--------------------------------
 2    live8          работает
 4    ro             работает
```

2. Подключение к локальному гипервизору с предварительной настройкой переменной окружения:

```
virsh list --all
```

```
 ID   Имя            Состояние
--------------------------------
 2    live8          работает
 4    ro             работает
```

3. Создание удаленного подключения к гипервизору:

```
virsh --connect qemu+ssh://virt-su@10.81.186.94/system
```

```
Добро пожаловать в virsh - интерактивный терминал виртуализации.
Введите «help» для получения справки по командам
«quit», чтобы завершить работу и выйти.
virsh # list --all
 ID   Имя              Состояние
--------------------------------
 -    redos-8          выключен
```

где:

- `virt-su` — имя пользователя, имеющего доступ к гипервизору;
- `10.81.186.94` — IP-адрес или имя сервера виртуализации.

1.5.3 Подключение к гипервизору с помощью менеджера ВМ

Для создания нового подключения в графическом менеджере ВМ необходимо в основном меню выбрать «Файл» — «Добавить соединение...».

Будет открыта форма добавления нового подключения, где необходимо из выпадающего списка выбрать гипервизор, к которому требуется осуществить подключение.

Далее нужно установить флаг в поле «Подключиться к удаленному узлу с помощью SSH» и указать в активировавшихся полях имя пользователя и IP-адрес сервера виртуализации.

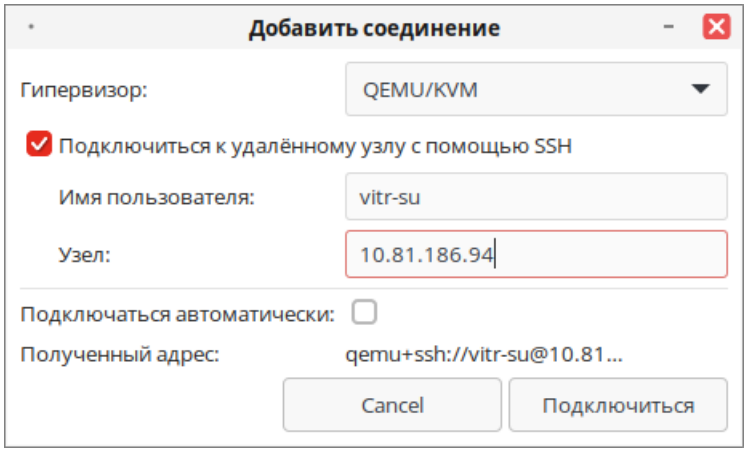

Рисунок 1.4 — Добавление нового подключения

При необходимости можно установить флаг для автоматического подключения к выбранному гипервизору.

Также открыть менеджер ВМ и подключиться к гипервизору можно из консоли, выполнив команду:

```
virt-manager -c qemu+ssh://virt-su@10.81.186.94/system
```

где:

- `virt-su` — имя пользователя, имеющего доступ к гипервизору;
- `10.81.186.94` — IP-адрес или имя сервера.

1.6 Создание виртуальных машин

Создание виртуальных машин является ключевым моментом при работе в среде виртуализации. Создание и установка ВМ может быть осуществлена как через терминал с помощью утилиты `virt-install`, так и через графический менеджер `virt-manager`.

1.6.1 Создание ВМ с помощью утилиты virt-install

Загрузите `ISO`-образ системы, которую необходимо установить. В примере будет использоваться образ РЕД ОС 8 с официального сайта.

После загрузки образа необходимо переместить его в доступный для `libvirt` каталог образов:

```
   cp /home/user/Загрузки/redos-8-20241206.4-Everything-x86_64-DVD1.iso \
/var/lib/libvirt/images/
```

После этого можно приступать к созданию ВМ. Для этого с правами администратора средства виртуализации (`superuser`) необходимо выполнить команду:

```
   virt-install --connect qemu:///system --name ro8-live --ram=1024 \
--disk pool=default,size=20,format=qcow2 --cdrom /var/lib/libvirt\
/images/redos-8-20241206.4-Everything-x86_64-DVD1.iso --vnc
```

где:

- `ro8` — имя ВМ;
- `1024` — объем ОЗУ;
- `pool=default` — настройка хранилища;
- `size=20` — объем выделенного пространства хранения (в ГБ);
- `format=qcow2` — формат хранилища;
- `/var/lib/libvirt/images/redos-8-20241206.4-Everything-x86_64-DVD1.iso` — путь к `ISO`-образу ОС;
- `--vnc` — запуск графического менеджера (на хостовой ОС должен быть установлен пакет `virt-viewer`).

Если все параметры заданы верно, будет открыта консоль с предложением установки гостевой ОС.

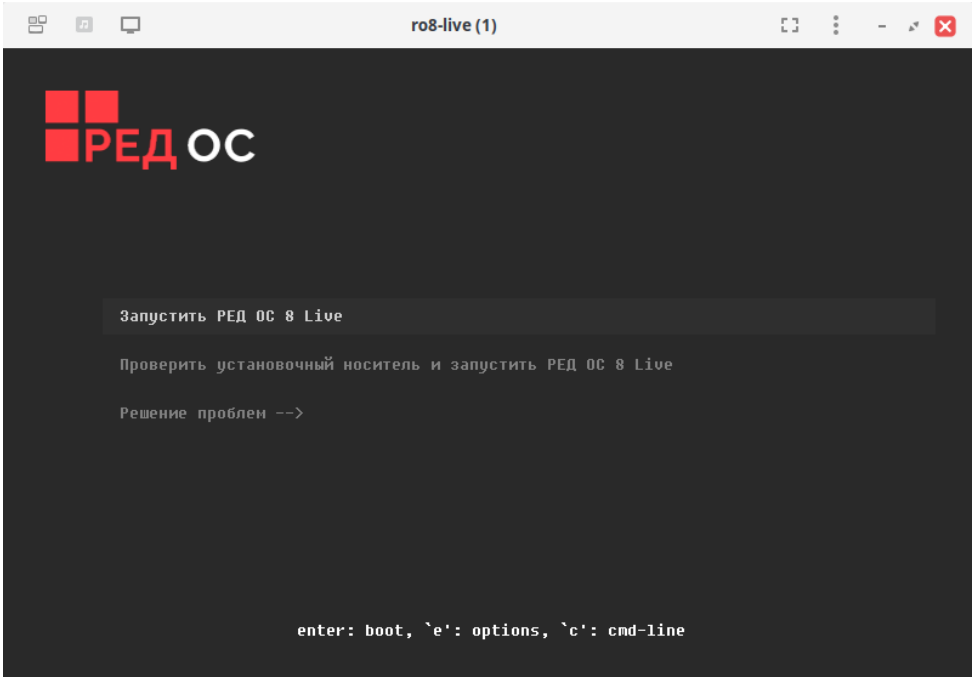

Рисунок 1.5 — Установка гостевой ОС

1.6.2 Создание ВМ с помощью менеджера virt-manager

Примечание. Необходимо обязательно подключиться к гипервизору, иначе в создании ВМ будет отказано и в окне «Новая виртуальная машина» будет отображено следующее сообщение об ошибке:

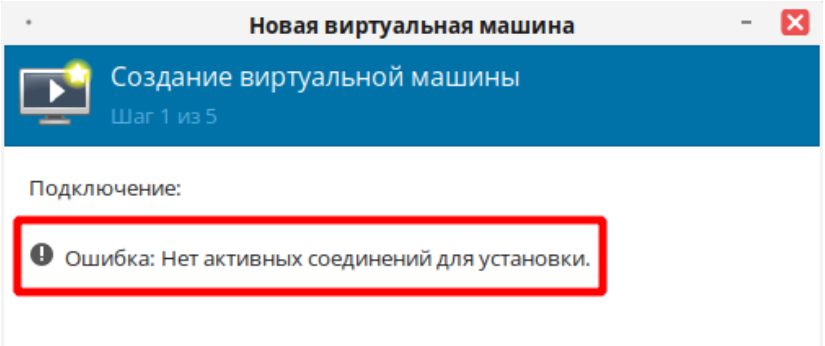

Рисунок 1.6 — Ошибка создания ВМ

Запустите менеджер ВМ через «Главное меню» — «Системные» — «Менеджер виртуальных машин» или через терминал командой:

```
virt-manager
```

Выполните подключение к гипервизору. Для этого в основном меню программы выберите «Файл» — «Добавить соединение. . . ».

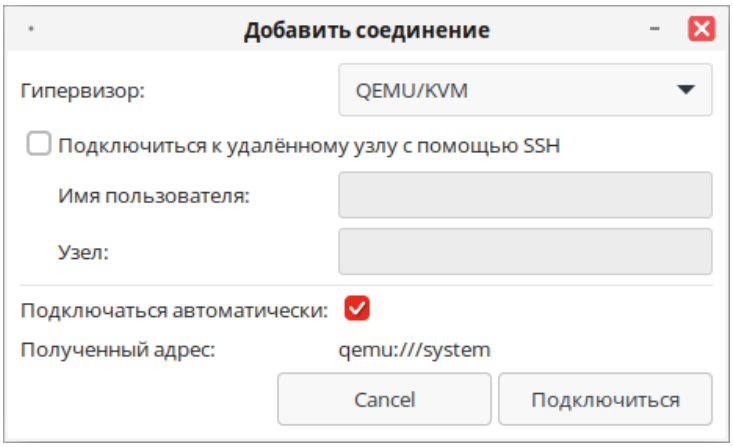

Рисунок 1.7 — Подключение к гипервизору

Выберите необходимый гипервизор и нажмите «Подключиться». После успешного подключения к гипервизору можно создавать ВМ. Для этого в панели инструментов нажмите на кнопку + или выберите «Файл» — «Создать виртуальную машину».

В открывшемся окне «Новая виртуальная машина» необходимо выбрать метод установки ОС — локальный или по сети.

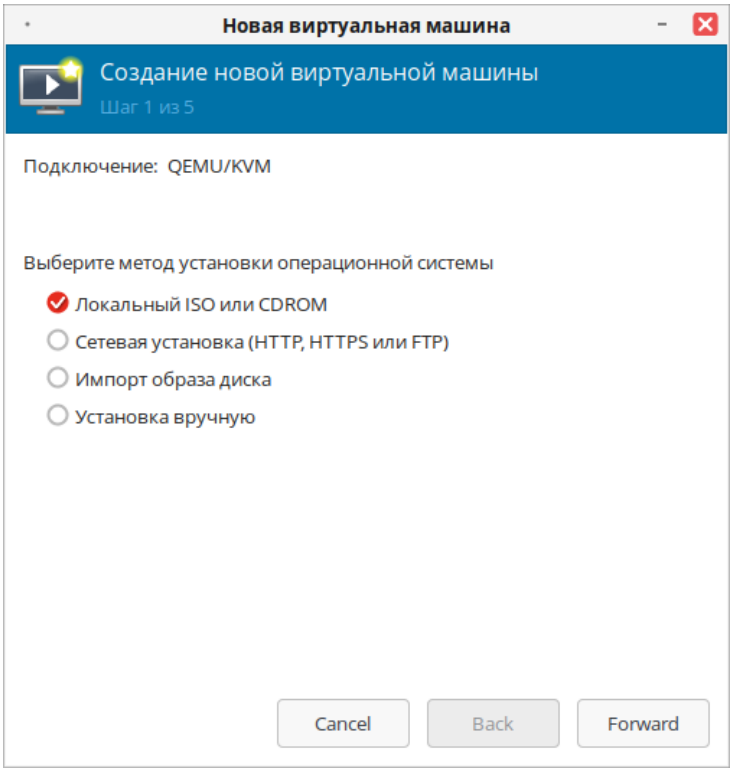

Рисунок 1.8 — Выбор метода установки новой ВМ

Далее необходимо указать путь к ISO-образу устанавливаемой системы, нажав кнопку «Обзор...».

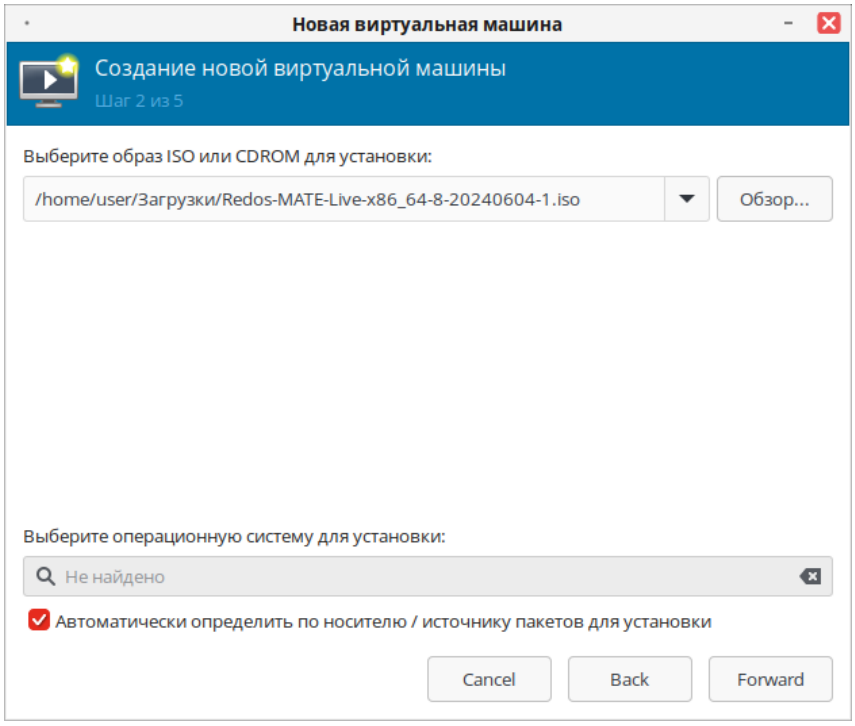

Рисунок 1.9 — Выбор типа ОС и ISO-образа

Снимите флаг в поле «Автоматически определить по носителю/источнику пакетов для установки» и в строке выбора типа ОС начните вводить ее название (в примере `redos`), будет выведен список доступных ОС, из которых потребуется выбрать нужную систему. Если нужной ОС нет в списке, выберите стандартную конфигурацию — `Generic`.

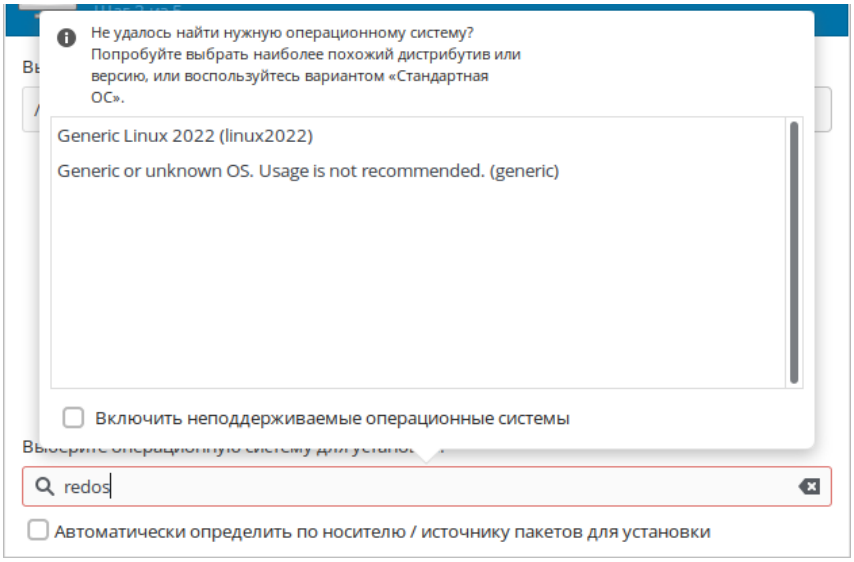

Рисунок 1.10 — Выбор стандартной конфигурации

Определите оптимальное количество ОЗУ и процессоров, исходя из доступных данных. Виртуальной машине понадобится достаточный для ее работы объем оперативной

памяти (как минимум 512 Мбайт). Стоит помнить, что ВМ используют физическую память. Выполнение слишком большого числа гостевых ОС или предоставление ОС недостаточного объема памяти может привести к повышенному использованию виртуальной памяти и области подкачки.

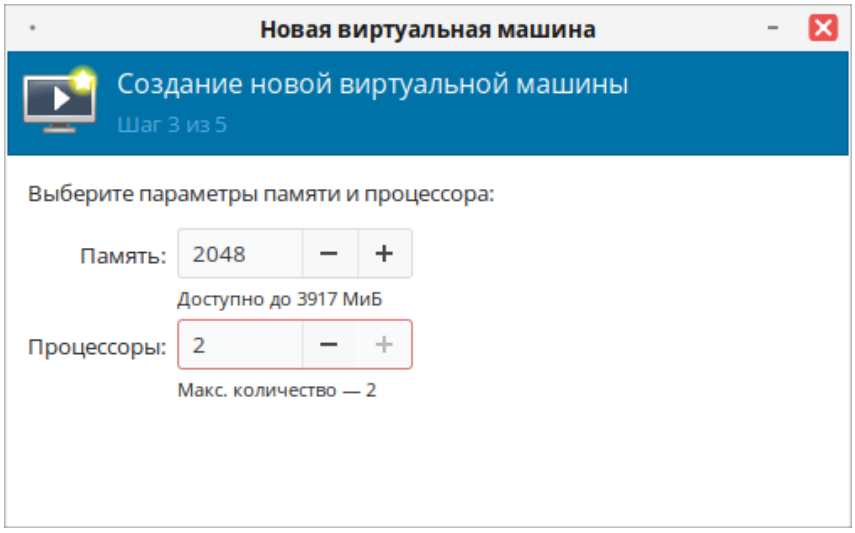

Рисунок 1.11 — Определение ОЗУ и процессоров

Далее необходимо настроить хранилище данных. Объем хранилища должен превышать минимально необходимый объем памяти, требуемый для установки системы.

Также, при желании, можно выделить дополнительное пространство для гостевой ОС.

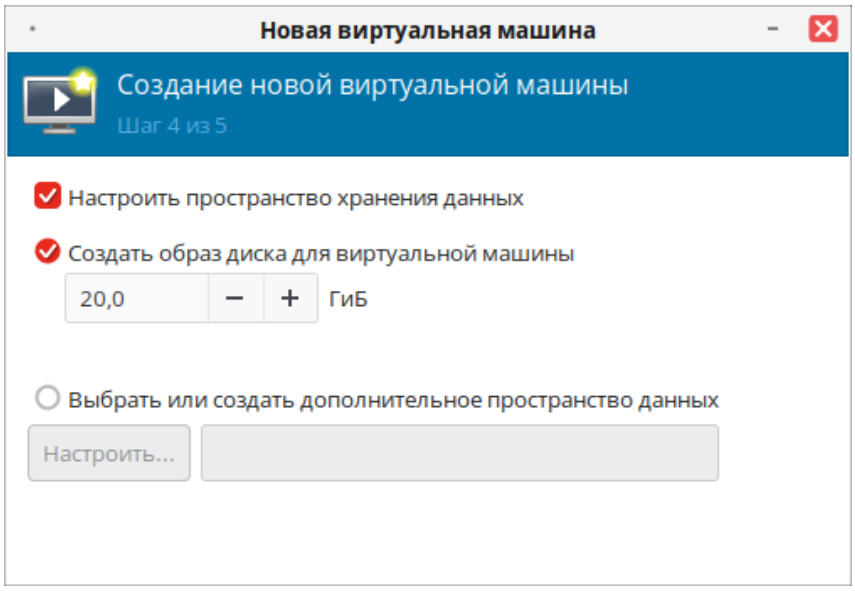

Рисунок 1.12 — Настройка пространства хранения данных

Проверьте правильность всех настроенных параметров. Если необходимо внести изменения, нажмите кнопку «Назад». Если все настроено верно, нажмите кнопку «Готово».

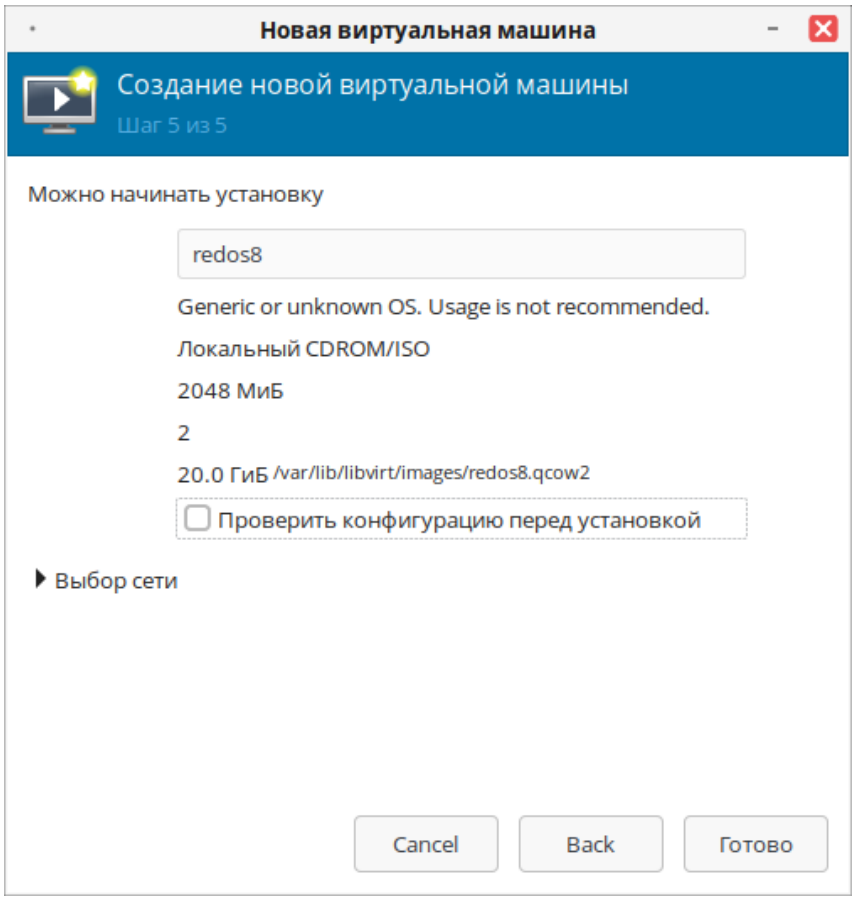

Рисунок 1.13 — Проверка параметров

Будет запущен процесс создания ВМ.

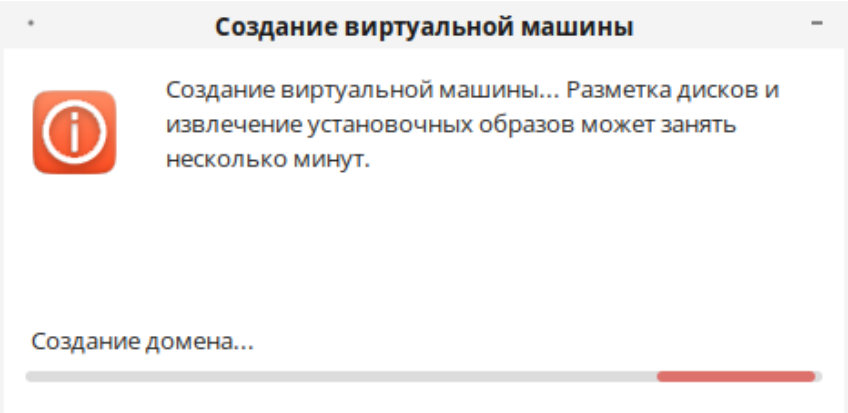

Рисунок 1.14 — Процесс создания ВМ

После успешного создания ВМ будет открыта консоль, в которой можно приступить к стандартной установке ОС.

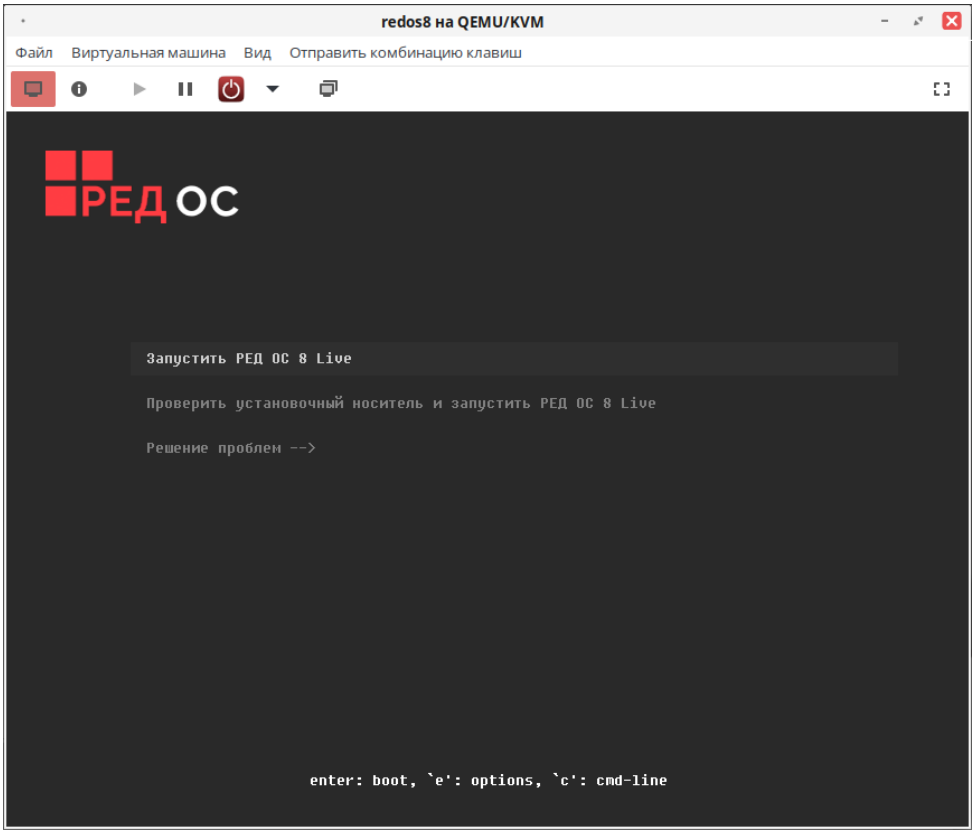

Рисунок 1.15 — Консоль установленной ВМ

1.6.3 Создание ВМ через файл конфигурации

Создание виртуальных машин возможно также с использованием файлов конфигурации формата XML.

За основу можно взять XML-файл уже существующей ВМ или создать собственный файл со своими параметрами.

Готовый XML-файл существующей ВМ можно просмотреть, выполнив команду вида:

```
virsh dumpxml <имя_ВМ>
```

Например:

```
virsh dumpxml redos8
```

```
<domain type='kvm' id='1'>
  <name>redos8</name>
  <uuid>52dba36c-4180-4ea3-9fe1-8573539653fa</uuid>
  <memory unit='KiB'>2097152</memory>
  <currentMemory unit='KiB'>2097152</currentMemory>
  <vcpu placement='static'>2</vcpu>
  <resource>
```

```
      <partition>/machine</partition>
    </resource>
    <os>
      <type arch='x86_64' machine='pc-i440fx-8.2'>hvm</type>
      <boot dev='cdrom'/>
      <boot dev='hd'/>
    </os>
    <features>
      <acpi/>
      <apic/>
      <vmport state='off'/>
    </features>
    <cpu mode='host-passthrough' check='none' migratable='on'/>
    <clock offset='utc'>
      <timer name='rtc' tickpolicy='catchup'/>
      <timer name='pit' tickpolicy='delay'/>
      <timer name='hpet' present='no'/>
    </clock>
    <on_poweroff>destroy</on_poweroff>
    <on_reboot>destroy</on_reboot>
    <on_crash>destroy</on_crash>
    <pm>
      <suspend-to-mem enabled='no'/>
      <suspend-to-disk enabled='no'/>
    </pm>
    <devices>
      <emulator>/usr/bin/qemu-system-x86_64</emulator>
      <disk type='file' device='disk'>
        <driver name='qemu' type='qcow2' discard='unmap'/>
        <source file='/var/lib/libvirt/images/redos8.qcow2' index='2'/>
        <backingStore/>
        <target dev='hda' bus='ide'/>
        <alias name='ide0-0-0'/>
        <address type='drive' controller='0' bus='0' target='0'
unit='0'/>
      </disk>
      <disk type='file' device='cdrom'>
        <driver name='qemu' type='raw'/>
        <source file='/home/user/Загрузки/Redos-MATE-Live-x86_64-8-
20240604-1.iso' index='1'/>
        <backingStore/>
        <target dev='hdb' bus='ide'/>
        <readonly/>
        <alias name='ide0-0-1'/>
        <address type='drive' controller='0' bus='0' target='0'
unit='1'/>
      </disk>
```

```
      <controller type='usb' index='0' model='ich9-ehci1'>
        <alias name='usb'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x7'/>
      </controller>
      <controller type='usb' index='0' model='ich9-uhci1'>
        <alias name='usb'/>
        <master startport='0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x0' multifunction='on'/>
      </controller>
      <controller type='usb' index='0' model='ich9-uhci2'>
        <alias name='usb'/>
        <master startport='2'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x1'/>
      </controller>
      <controller type='usb' index='0' model='ich9-uhci3'>
        <alias name='usb'/>
        <master startport='4'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x2'/>
      </controller>
      <controller type='pci' index='0' model='pci-root'>
        <alias name='pci.0'/>
      </controller>
      <controller type='ide' index='0'>
        <alias name='ide'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x01'
function='0x1'/>
      </controller>
      <controller type='virtio-serial' index='0'>
        <alias name='virtio-serial0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x06'
function='0x0'/>
      </controller>
      <interface type='network'>
        <mac address='52:54:00:50:d4:2a'/>
        <source network='default' portid='eddeba22-dbad-40cf-a91b-
0c9f0cd62149' bridge='virbr0'/>
        <target dev='vnet0'/>
        <model type='e1000'/>
        <alias name='net0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0'/>
      </interface>
      <serial type='pty'>
```

```
        <source path='/dev/pts/1'/>
        <target type='isa-serial' port='0'>
          <model name='isa-serial'/>
        </target>
        <alias name='serial0'/>
      </serial>
      <console type='pty' tty='/dev/pts/1'>
        <source path='/dev/pts/1'/>
        <target type='serial' port='0'/>
        <alias name='serial0'/>
      </console>
      <channel type='spicevmc'>
        <target type='virtio' name='com.redhat.spice.0' state='discon-
nected'/>
        <alias name='channel0'/>
        <address type='virtio-serial' controller='0' bus='0' port='1'/>
      </channel>
      <input type='tablet' bus='usb'>
        <alias name='input0'/>
        <address type='usb' bus='0' port='1'/>
      </input>
      <input type='mouse' bus='ps2'>
        <alias name='input1'/>
      </input>
      <input type='keyboard' bus='ps2'>
        <alias name='input2'/>
      </input>
      <graphics type='spice' port='5900' autoport='yes' listen=
'127.0.0.1'>
        <listen type='address' address='127.0.0.1'/>
        <image compression='off'/>
      </graphics>
      <sound model='ich6'>
        <alias name='sound0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x04'
function='0x0'/>
      </sound>
      <audio id='1' type='spice'/>
      <video>
        <model type='qxl' ram='65536' vram='65536' vgamem='16384'
heads='1' primary='yes'/>
        <alias name='video0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x02'
function='0x0'/>
      </video>
      <redirdev bus='usb' type='spicevmc'>
        <alias name='redir0'/>
```

```
        <address type='usb' bus='0' port='2'/>
      </redirdev>
      <redirdev bus='usb' type='spicevmc'>
        <alias name='redir1'/>
        <address type='usb' bus='0' port='3'/>
      </redirdev>
      <memballoon model='virtio'>
        <alias name='balloon0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x07'
function='0x0'/>
      </memballoon>
    </devices>
    <seclabel type='dynamic' model='selinux' relabel='yes'>
      <label>system_u:system_r:svirt_t:s0:c606,c747</label>
      <imagelabel>system_u:object_r:svirt_image_t:s0:c606,c747</image-
label>
    </seclabel>
    <seclabel type='dynamic' model='dac' relabel='yes'>
      <label>+107:+107</label>
      <imagelabel>+107:+107</imagelabel>
    </seclabel>
  </domain>
```

Как уже рассматривалось ранее (см. п. 1.2.3 «Утилита virsh»), данная команда выводит `XML`-файл на стандартный вывод (`stdout`). Для сохранения вывода в отдельный файл, выполните команду:

```
virsh dumpxml redos8 > redos8_config.xml
```

Для внесения каких-либо изменений в файл конфигурации выполните команду:

```
virsh edit redos8_config.xml
```

Для создания ВМ из `XML`-файла выполните команду:

```
virsh create redos8_config.xml
```

1.7 Управление ВМ

Управление ВМ подразумевает такие действия, как:

- мониторинг состояния и ресурсов ВМ;
- управление питанием ВМ;
- создание снимков гостевой системы.

1.7.1 Управление ВМ с помощью утилиты virsh

Запуск ВМ

Для запуска ВМ выполните команду вида:

```
virsh start <ID_домена, имя_домена, UUID_домена>
```

Приостановка ВМ

Для приостановки ВМ выполните команду вида:

```
virsh suspend <ID_домена, имя_домена, UUID_домена>
```

В этом состоянии виртуальная машина все еще продолжает использовать системную память, но освобождает ресурсы процессора. Операции ввода и вывода приостанавливаются. Работа виртуальной машины может быть возобновлена с помощью команды `resume`.

Возобновление работы ВМ

Для возобновления работы ВМ выполните команду вида:

```
virsh resume <ID_домена, имя_домена, UUID_домена>
```

Работа машины будет возобновлена немедленно.

Сохранение состояния ВМ

Для сохранения текущего состояния ВМ выполните команду:

```
virsh save <ID_домена, имя_домена, UUID_домена> <файл>
```

В результате выполнения данной команды виртуальная машина будет остановлена, а ее данные будут сохранены в заданный файл (это может занять некоторое время в зависимости от доступного гостевой ОС объема памяти). Состояние может быть позднее восстановлено с помощью команды `restore`.

Восстановление виртуальной машины

Чтобы восстановить виртуальную машину, ранее сохраненную с помощью команды `virsh save`, выполните команду:

```
virsh restore <файл>
```

Сохраненная машина будет восстановлена из файла и перезапущена, что может занять некоторое время. Имя и идентификатор `UUID` виртуальной машины останутся неизменными, но будет предоставлен новый идентификатор домена.

Создание снимка ВМ

Для создания снимка (снапшота) ВМ выполните команду:

```
virsh snapshot-create <ID_домена, имя_домена, UUID_домена>
```

Снимок диска или памяти будет успешно создан. Позднее состояние ВМ можно будет востановить из созданного снапшота командой `snapshot-revert`.

Восстановление ВМ из снимка (снапшота)

Для восстановления состояния ВМ из снапшота выполните команду:

```
virsh snapshot-revert <ID_домена, имя_домена, UUID_домена>
[--snapshotname <имя_снимка>]
```

Снимок будет восстановлен и состояние ВМ будет возвращено к указанному или

текущему (последнему) снапшоту.

Завершение работы виртуальной машины

Для завершения работы ВМ выполните:

```
virsh shutdown <ID_домена, имя_домена, UUID_домена>
```

Поведение выключаемой гостевой ОС можно контролировать с помощью параметра `on_shutdown` в ее файле конфигурации.

Перезагрузка виртуальной машины

Для перезагрузки ВМ выполните:

```
virsh reboot <ID_домена, имя_домена, UUID_домена>
```

Поведение перезагружаемой гостевой ОС можно контролировать с помощью параметра `on_reboot` в ее файле конфигурации.

Принудительная остановка виртуальной машины

Для принудительной остановки ВМ выполните:

```
virsh destroy <ID_домена, имя_домена, UUID_домена>
```

Эта команда выполнит немедленное отключение и остановит все сессии гостевых доменов.

> **Важно!** Такое внезапное завершение может привести к повреждению файловых систем виртуальной машины. Команду `destroy` рекомендуется использовать только в случае, если виртуальная машина не отвечает на запросы. ■

Удаление ВМ

Перед удалением ВМ должна быть выключена командой `shutdown` либо `destroy`.

Также не может быть удалена ВМ, которая имеет снапшоты. Проверить наличие снапшотов у удаляемой ВМ можно командой:

```
virsh snapshot-list --domain <имя_домена>
```

Если снапшоты присутствуют, сначала потребуется удалить их. Для этого выполните команду:

```
virsh snapshot-delete <имя_домена> --snapshotname <имя_снапшота>
--children
```

Снапшот будет удален вместе с его дочерними элементами (параметр `--children`).

После этого удалите ВМ (для удаления всех принадлежащих домену томов хранения данных используйте параметр `--remove-all-storage`):

```
virsh undefine <имя_домена> --remove-all-storage
```

ВМ и используемые тома будут удалены.

Для безопасного удаления данных с томов хранения ВМ без возможности их дальнейшего восстановления можно использовать дополнительный параметр `--wipe-storage`. Команда очистки данных хранилища и полного удаления ВМ выглядит следующим образом:

```
undefine <имя_домена> --wipe-storage --remove-all-storage
```

ВМ и все данные будут полностью удалены.

Создание томов хранения для ВМ

Для создания нового тома хранения выполните команду:

```
virsh vol-create-as <имя_пула> <имя_тома> <размер_тома_в_байтах>
[--format <строка>]
```

где:

- `<имя_пула>` — пул, в котором необходимо создать новый том;
- `<имя_тома>` — имя нового тома хранения;
- `<размер_тома_в_байтах>` — размер нового тома в байтах;
- `[--format <строка>]` — формат нового тома хранения: `raw`, `bochs`, `qcow`, `qcow2`, `qed`, `vmdk`.

Новый том хранения будет успешно создан. Размер нового тома хранения не должен превышать объем доступного свободного пространства на хостовой машине.

Список доступных томов хранения

Для вывода списка доступных томов хранения используется команда:

```
virsh vol-list <имя_пула>
```

Например:

```
virsh vol-list default
```

```
 Имя                  Путь
----------------------------------------------------------------------
 redos-...-8.iso      /var/lib/libvirt/images/redos-...-8.iso
 ro-test.qcow2        /var/lib/libvirt/images/ro-test.qcow2
 redos8.qcow2         /var/lib/libvirt/images/redos8.qcow2
 ro8.qcow2            /var/lib/libvirt/images/ro8.qcow
```

Изменение размера тома хранения

Для изменения размера тома хранения выполните команду:

```
virsh vol-resize <имя_тома> <размер_тома>
```

где

- `<имя_тома>` — том, для которого необходимо изменить размер;
- `<размер_тома>` — новый размер тома, по умолчанию в байтах.

Безопасно использовать только для томов, которые не используются активной гостевой системой. При добавлении параметра `--delta` установленный размер тома будет увеличен (или уменьшен) на указанный размер. Например:

```
virsh vol-resize /var/lib/libvirt/images/ro8.qcow2 1G --delta
```

```
Размер тома «redos8.qcow2» успешно изменён на 1G
```

В примере размер указанного тома будет увеличен на 1ГБ.

Сохранение состояния жесткого диска ВМ

Для сохранения содержимого тома хранения ВМ в какой-либо локальный файл выполните команду:

```
virsh vol-download <имя_тома> <имя_файла>
```

Данные тома хранения будут записаны в указанный файл, который впоследствии можно скопировать и восстановить на другой ВМ или на этой же ВМ.

Восстановление состояния жесткого диска ВМ

Для восстановления содержимого тома хранения ВМ из файла выполните команду:

```
virsh vol-upload <имя_тома> <имя_файла>
```

Данные тома хранения будут восстановлены.

Определение идентификатора домена

Для определения идентификатора домена виртуальной машины выполните:

```
virsh domid <имя_домена, UUID_домена>
```

Определение имени домена

Для определения имени домена виртуальной машины выполните:

```
virsh domname <ID_домена, UUID_домена>
```

Определение UUID

Для определения универсального идентификатора `UUID` виртуальной машины выполните:

```
virsh domuuid <ID_домена, имя_домена>
```

Пример вывода команды:

```
virsh domuuid redos8
```

```
52dba36c-4180-4ea3-9fe1-8573539653fa
```

Получение информации о виртуальной машине

Для получения информации о ВМ выполните:

```
virsh dominfo <ID_домена, имя_домена, UUID_домена>
```

Например:

```
virsh dominfo redos8
```

```
ID:                -
Имя:               redos8
UUID:              52dba36c-4180-4ea3-9fe1-8573539653fa
Тип ОС:            hvm
Состояние:         выключен
CPU:               2
Макс.память:       2097152 KiB
Занято памяти:     2097152 KiB
Постоянство:       yes
Автозапуск:        выкл.
Управляемое сохранение:   no
Модель безопасности:      selinux
DOI безопасности:         0
```

Получение информации об узле

Для получения информации об узле выполните:

```
virsh nodeinfo
```

Например:

```
virsh nodeinfo
```

```
Модель процессора:    x86_64
CPU:                  2
Частота процессора:   3292 MHz
Сокеты:               2
Ядер на сокет:        1
Потоков на ядро:      1
Ячейки NUMA:          1
Объём памяти:         4011160 Ki
```

Просмотр списка виртуальных машин

Для просмотра списка виртуальных машин и их состояния выполните:

```
virsh list
```

Можно добавить аргументы:

- `--inactive` — покажет список неактивных доменов (неактивным считается тот домен, который был определен, но в настоящий момент не является активным).
- `--all` — покажет все виртуальные машины независимо от их состояния.

Например:

```
virsh list --all
```

```
 ID    Имя        Состояние
---------------------------
 2     ro8        работает
 3     ro8-live   работает
 -     redos8     выключ
```

Столбец «Состояние» может содержать следующие значения:

- `работает` — работающие виртуальные машины, т. е. те машины, которые используют ресурсы процессора в момент выполнения команды;
- `заблокирован` — заблокированные, неработающие машины. Такой статус может быть вызван ожиданием ввода/вывода или пребыванием машины в спящем режиме;
- `приостановлен` — приостановленные домены. Данное состояние наступает, когда администратор нажал кнопку паузы в окне менеджера виртуальных машин или выполнил команду `virsh suspend`. В приостановленном состоянии гостевая ОС продолжает потреблять ресурсы, но не может занимать больше процессорных ресурсов;
- `выключен` — виртуальные машины, завершающие свою работу. При получении виртуальной машиной сигнала завершения работы, она начнет завершать все процессы. Стоит отметить, что некоторые операционные системы не отвечают на такие сигналы;
- `dying` — сбойные домены и домены, которые не смогли корректно завершить свою работу;
- `crashed` — сбойные домены, работа которых была прервана. В этом состоянии домены находятся, если не была настроена их перезагрузка в случае сбоя.

Получение информации о виртуальных процессорах

Для получения информации о виртуальных процессорах выполните:

```
virsh vcpuinfo <ID_домена, имя_домена, UUID_домена>
```

Пример вывода:

```
virsh vcpuinfo ro8
```

```
Виртуальный процессор:: 0
CPU:                    1
Состояние:              работает
Время CPU:              139,3s
Соответствие ЦП:        yy

Виртуальный процессор:: 1
CPU:                    1
Состояние:              работает
```

```
Время CPU:              70,1s
Соответствие ЦП:        yy
```

Изменение числа виртуальных процессоров

Для изменения числа процессоров для домена выполните:

```
virsh setvcpus <ID_домена, имя_домена, UUID_домена> <число>
```

Обратите внимание, что заданное число не может превышать значение, определенное при создании гостевой ОС.

Изменение выделенного объема памяти

Для изменения выделенного виртуальной машине объема памяти выполните:

```
virsh setmem <ID_домена, имя_домена> <число>
```

Объем памяти, определяемый заданным числом, должен быть указан в килобайтах. Обратите внимание, что объем не может превышать значение, определенное при создании виртуальной машины, но в то же время не должен быть меньше 64 МБ. Изменение максимального объема памяти может оказать влияние на функциональность ВМ только в том случае, если указанный размер меньше исходного. В таком случае использование памяти будет ограничено.

Получение информации о блочных устройствах

Для получения информации о блочных устройствах работающей виртуальной машины выполните:

```
virsh domblkstat <виртуальная_машина> <блочное_устройство>
```

Получение информации о сетевых устройствах

Для получения информации о сетевых интерфейсах работающей виртуальной машины выполните:

```
virsh domifstat <виртуальная_машина> <интерфейс>
```

Управление виртуальными сетями

Для просмотра списка виртуальных сетей выполните:

```
virsh net-list
```

Пример вывода этой команды:

```
virsh net-list
```

```
 Имя       Состояние    Автозапуск    Постоянный
----------------------------------------------------------------
 default   активен      yes           yes
```

Просмотреть информацию для заданной виртуальной сети можно командой:

```
virsh net-dumpxml <имя_сети>
```

Пример вывода этой команды (в формате XML):

```
virsh net-dumpxml default
```

```
<network connections='3'>
  <name>default</name>
  <uuid>31e63413-2ca4-4a66-8b5e-5c8268aaad41</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:bd:26:eb'/>
  <ip address='192.168.124.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.124.2' end='192.168.124.254'/>
    </dhcp>
  </ip>
</network>
```

Другие команды управления виртуальными сетями:

- `virsh net-autostart <имя_сети>` — автоматический запуск заданной сети;
- `virsh net-create <файл_XML>` — создание и запуск новой сети на основании существующего XML-файла;
- `virsh net-define <файл_XML>` — создание нового сетевого устройства на основании существующего XML-файла. Устройство не будет запущено;
- `virsh net-destroy <имя_сети>` — удаление заданной сети;
- `virsh net-name <UUID_сети>` — преобразование заданного идентификатора в имя сети;
- `virsh net-uuid <имя_сети>` — преобразование заданного имени в идентификатор UUID;
- `virsh net-start <имя_неактивной_сети>` — запуск неактивной сети;
- `virsh net-undefine <имя_неактивной_сети>` — удаление определения неактивной сети.

1.7.2 Управление ВМ с помощью менеджера virt-manager

Менеджер `virt-manager` предоставляет графический интерфейс для доступа к гипервизорам и виртуальным машинам в локальной и удаленных системах. Кроме того, `virt-manager` выполняет управляющие функции:

- выделение памяти;
- выделение виртуальных процессоров;
- мониторинг производительности;
- сохранение и восстановление, приостановка и возобновление работы, запуск и завершение работы виртуальных машин;
- доступ к текстовой и графической консоли.

Производительность ВМ

Для просмотра графиков производительности какой-либо ВМ дважды нажмите ЛКМ по имени нужной ВМ, будут открыты подробные сведения.

На вкладке «Производительность» отображаются диаграммы и статистика использования ресурсов в реальном времени.

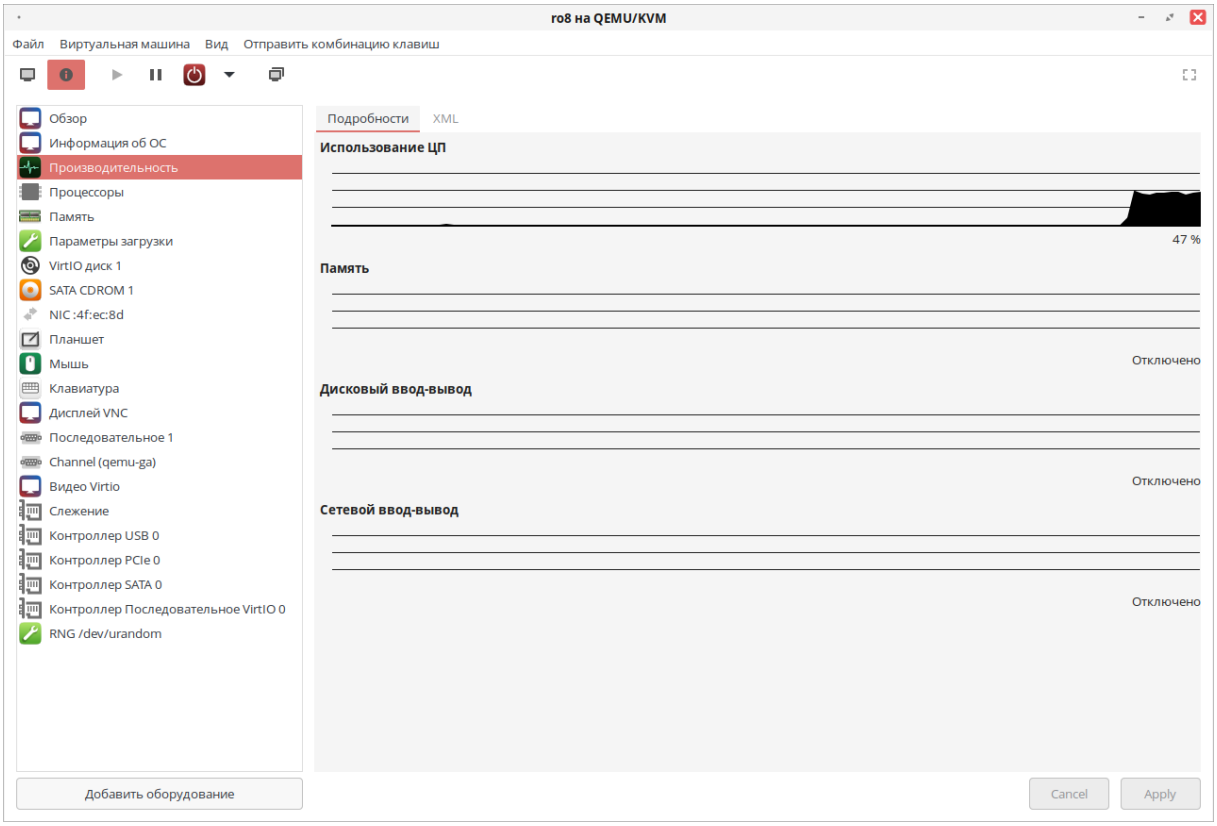

Рисунок 1.16 — Просмотр графиков производительности ВМ

Графическая консоль виртуальной машины

Окно содержит графическую консоль виртуальной машины. По умолчанию используется протокол `SPICE` для обеспечения доступа к консоли. Доступ к графической консоли ВМ имеет только пользователь с правами администратора ВМ.

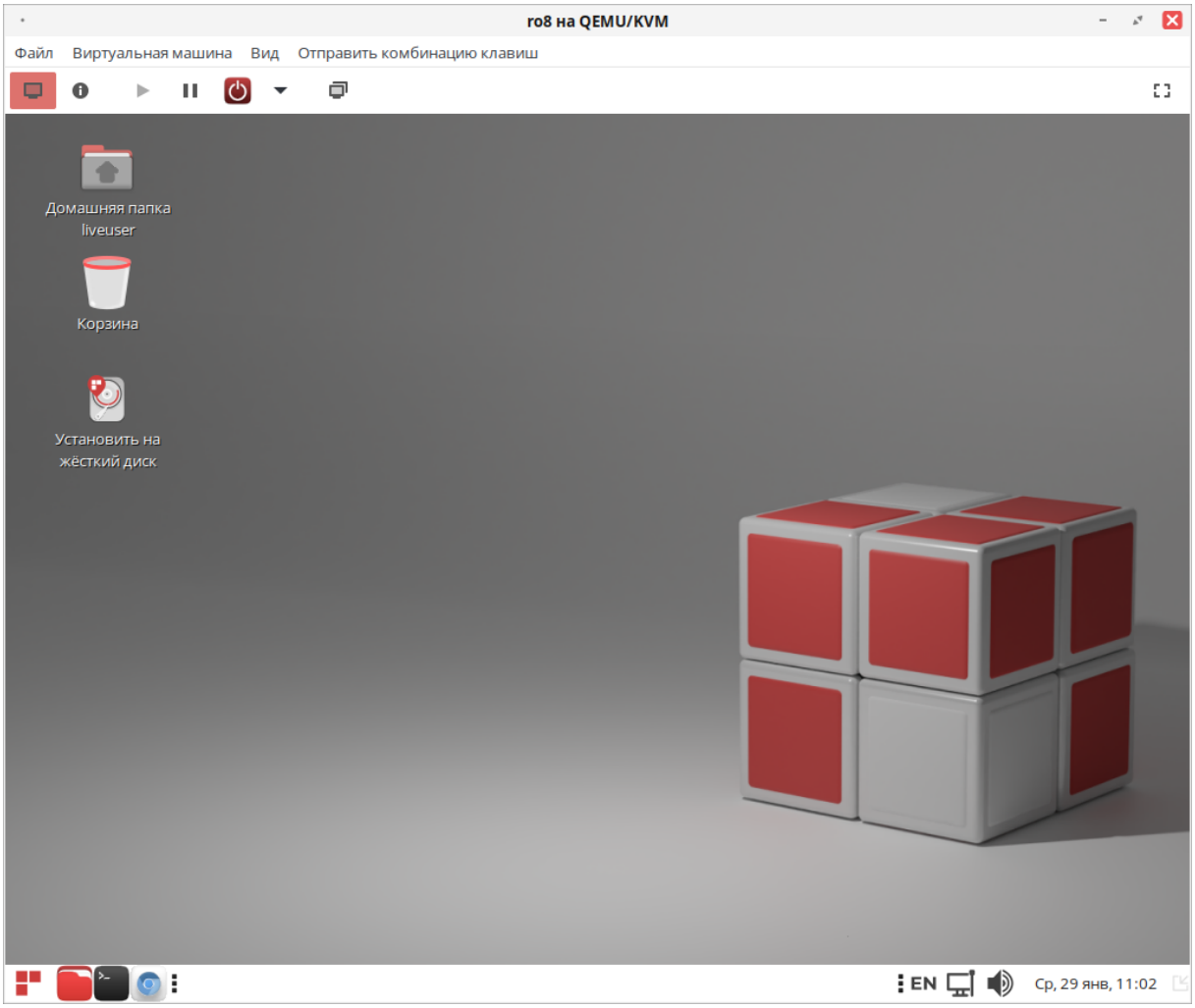

Рисунок 1.17 — Графическая консоль ВМ

Окружение локального рабочего стола способно перехватывать комбинации клавиш (например, `Ctrl+Alt+F11`) для предотвращения их отправки гостевой машине. Чтобы отправлять такие последовательности, используйте свойство "западания" клавиш `virt-manager`. Нажмите функциональную клавишу (`Ctrl` или `Alt`) 3 раза для ее перехода в нажатое состояние. Клавиша будет считаться нажатой до тех пор, пока не будет нажата любая другая клавиша, отличная от модификатора.

Таким образом, чтобы передать гостевой системе комбинацию `Ctrl+Alt+F11`, необходимо последовательно нажать `Ctrl-Ctrl-Ctrl+Alt+F11`.

Просмотр информации о гостевой системе

С помощью менеджера виртуальных машин можно получить доступ к подробной информации обо всех виртуальных машинах. Для этого выполните следующий алгоритм действий:

1. В главном окне менеджера выберите виртуальную машину.

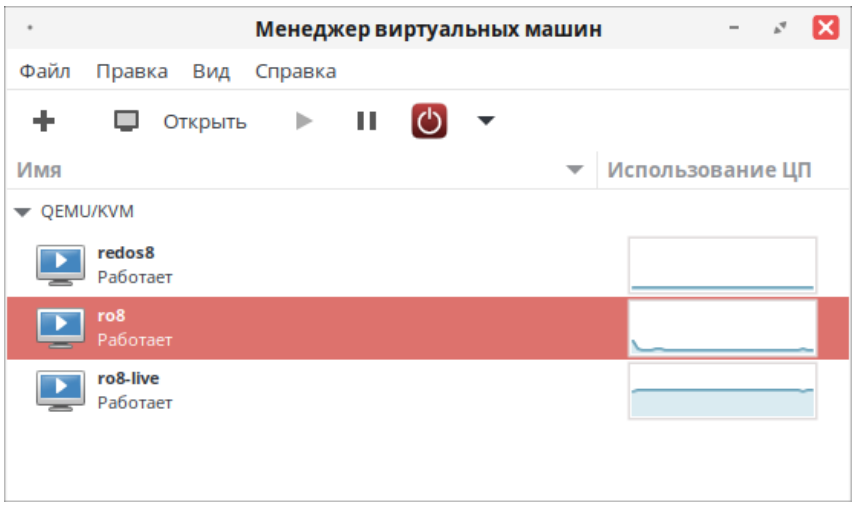

Рисунок 1.18 — Выбор ВМ

2. В пункте меню «Правка» выберите «Свойства виртуальной машины» или дважды нажмите ЛКМ по имени ВМ.

Будет выведено окно с подробными сведениями о виртуальной машине, на вкладке «Производительность» будет доступна информация об использовании ресурсов процессора и памяти.

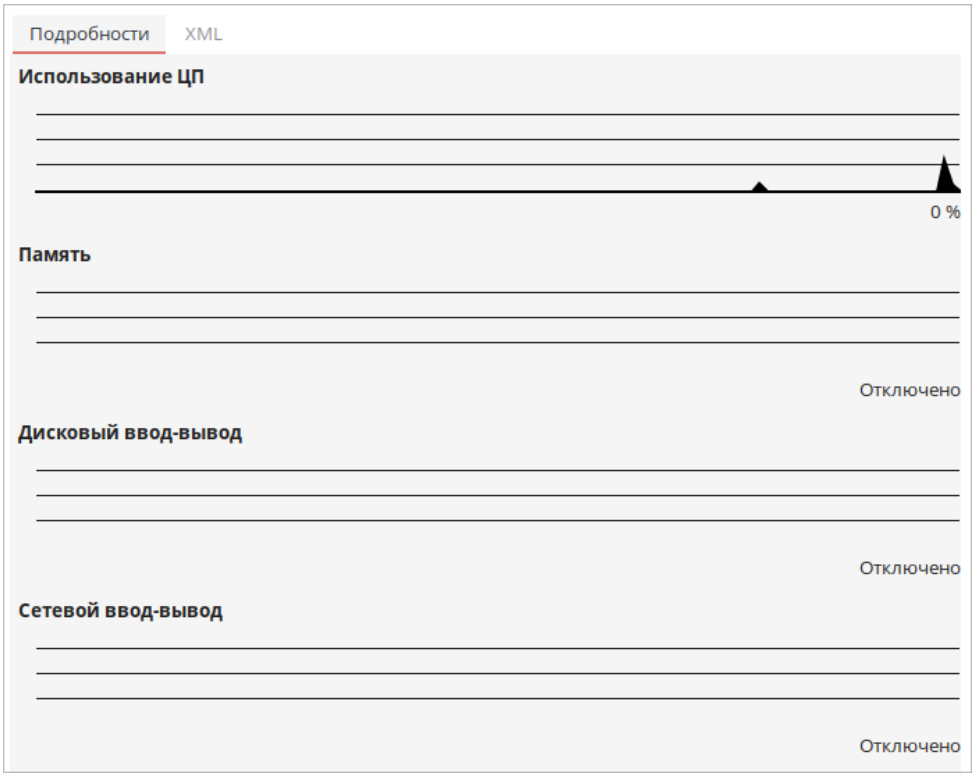

Рисунок 1.19 — Графики использования ресурсов процессора и памяти

3. Для просмотра или изменения числа виртуальных процессоров выберите «Процессоры».

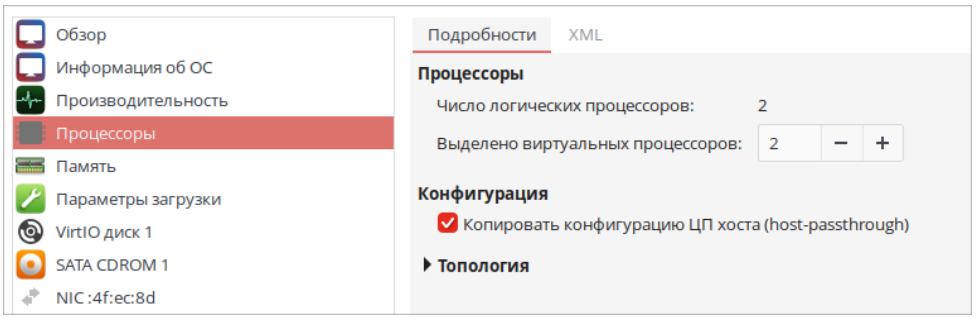

Рисунок 1.20 — Просмотр числа процессоров ВМ

4. Для просмотра или изменения распределения ресурсов памяти выберите «Память».

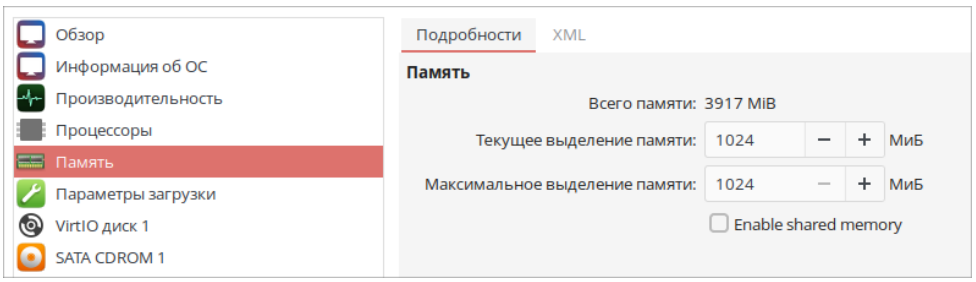

Рисунок 1.21 — Просмотр ресурсов ОЗУ

5. Для просмотра или изменения дисковой конфигурации выберите «Диск».

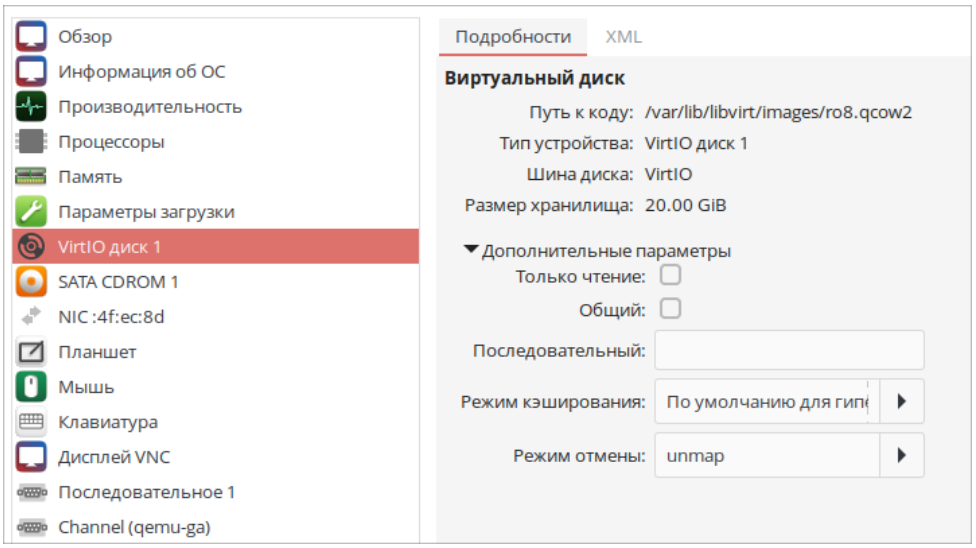

Рисунок 1.22 — Просмотр конфигурации диска

6. Для просмотра или изменения сетевой конфигурации выберите «NIC».

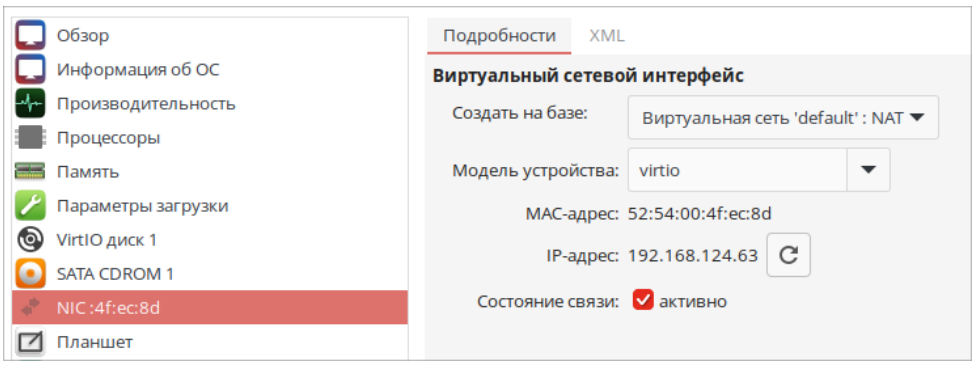

Рисунок 1.23 — Просмотр настроек сетевого интерфейса

Мониторинг состояния

С помощью менеджера ВМ можно редактировать настройки контроля статуса.

Для настройки мониторинга состояния и активации консолей в окне Менеджера ВМ в пункте меню «Правка» выберите «Preferences».

На вкладке «Статистика» укажите необходимый интервал обновления состояния ВМ.

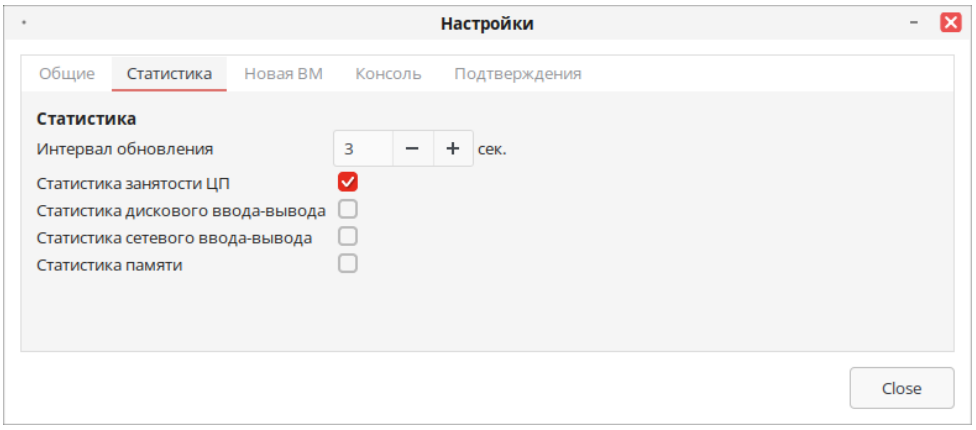

Рисунок 1.24 — Вкладка Статистика

На вкладке «Новая ВМ» выполните настройку графики, перенаправление USB, формат хранилища и ЦП.

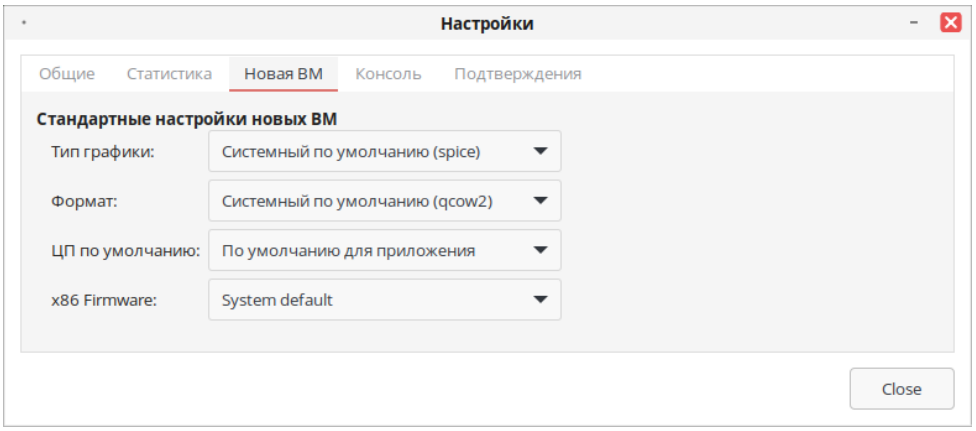

Рисунок 1.25 — Вкладка Новая ВМ

На вкладке «Консоль» можно настроить параметры графической консоли ВМ, такие как масштаб, разрешение окна, клавиши для освобождения курсора и пр.

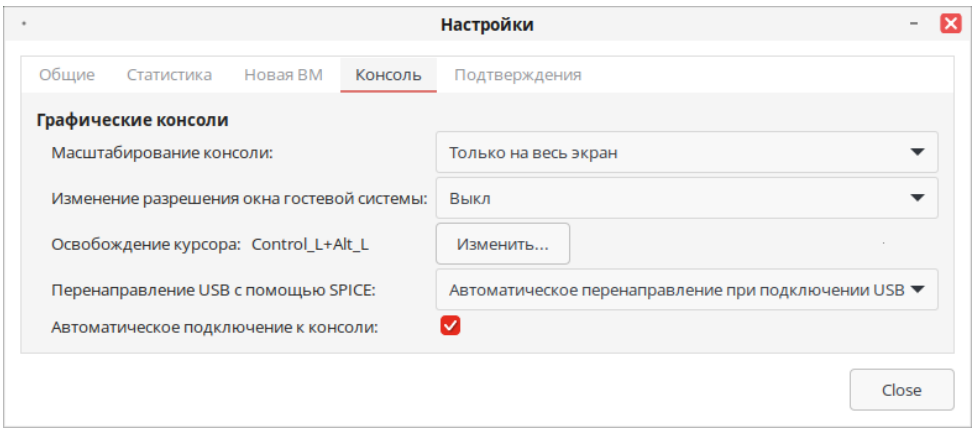

Рисунок 1.26 — Вкладка Консоль

На вкладке «Подтверждения» можно установить, на какие действия пользователя потребуется дополнительное подтверждение. Например, при выборе принудительного завершения сеанса будет выполнен дополнительный запрос, в котором пользователь может подтвердить данное действие или отказаться от него.

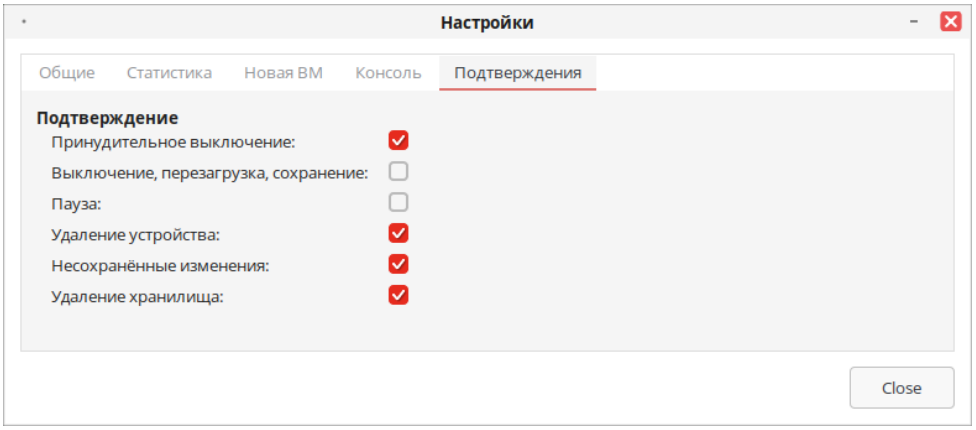

Рисунок 1.27 — Вкладка Подтверждения

Просмотр состояния виртуальной машины

Состояния ВМ отображаются в главном окне менеджера в списке созданных ВМ под их названием.

Состояния могут принимать следующие значения:

- `работает`;
- `выключена`;
- `пауза`;
- `сохранено`.

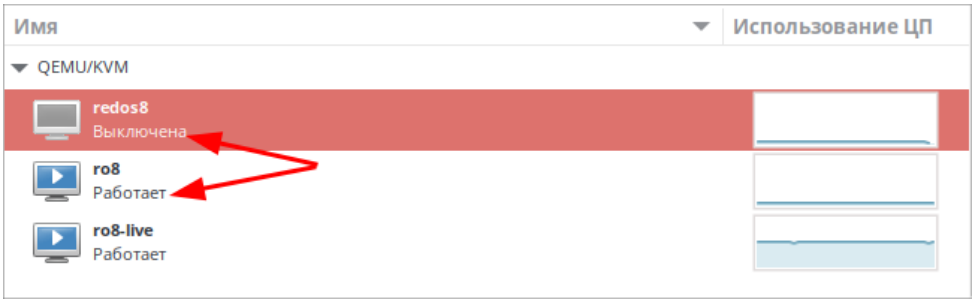

Рисунок 1.28 — Просмотр состояний ВМ

Управление виртуальной сетью

Для настройки виртуальных сетей необходимо:

1. В окне менеджера ВМ в пункте меню «Правка» выбрать «Свойства подключения».

2. Перейти на вкладку «Виртуальные сети».

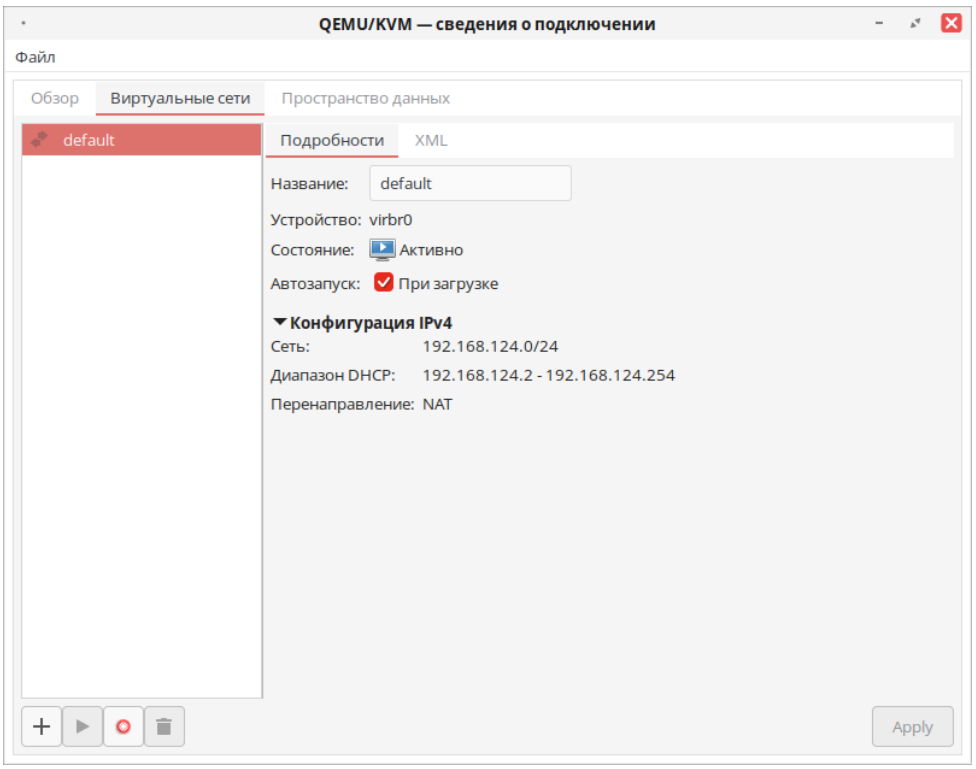

Рисунок 1.29 — Управление виртуальными сетями

3. Доступные виртуальные сети будут отображаться в левой части окна. Выберите сеть для доступа к ее настройкам.

4. Для сохранения внесенных изменений нажмите «Применить».

Создание виртуальной сети

Для создания новой виртуальной сети нажмите на кнопку + в левой части окна.

Будет открыто окно «Создание новой виртуальной сети», в котором необходимо указать название сети, выбрать из списка режим подключения и способ перенаправления сети.

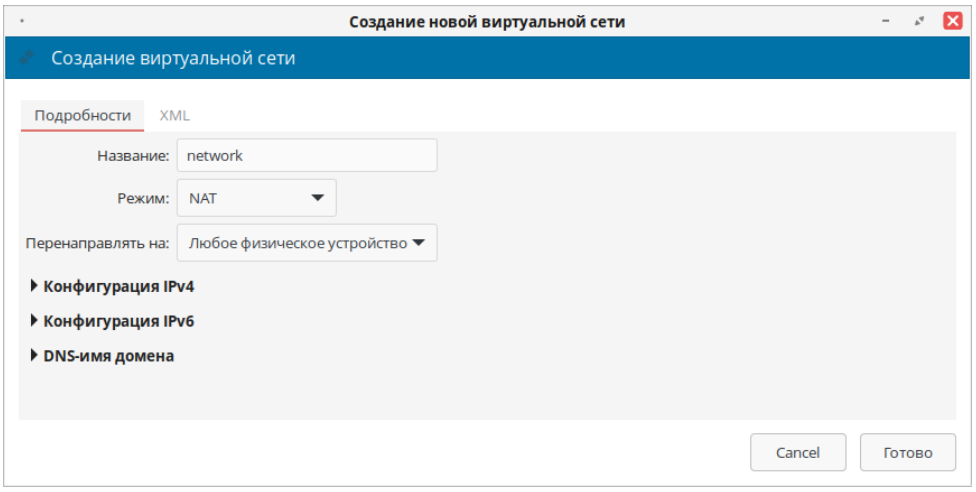

Рисунок 1.30 — Создание новой виртуальной сети

Далее потребуется ввести пространство адресов `IPv4` для виртуальной сети и указать диапазон `DHCP` для вашей виртуальной сети, задав начальный и конечный адрес.

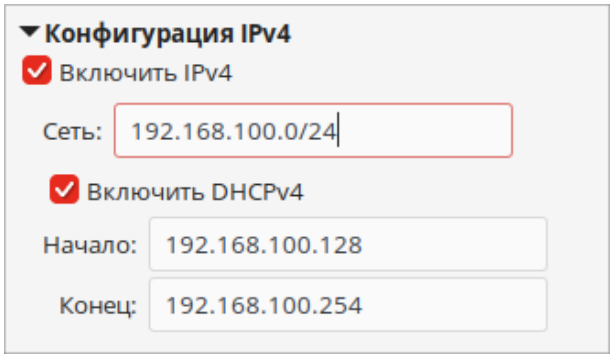

Рисунок 1.31 — Настройка пространства адресов IPv4

Проверьте правильность указанных данных и нажмите «Готово» для создания новой виртуальной сети.

Сведения о новой виртуальной сети можно получить на вкладке «Виртуальные сети» в окне «Свойства подключения».

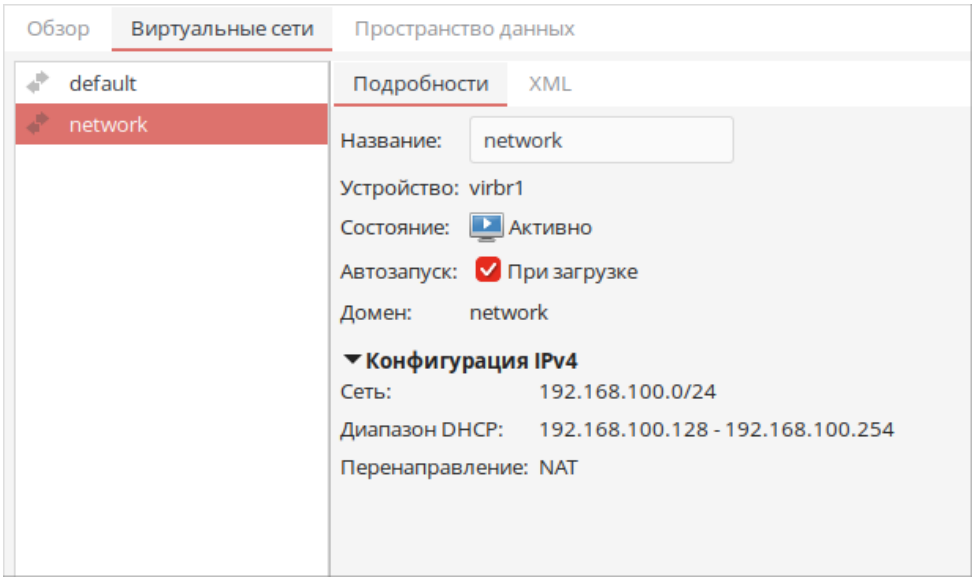

Рисунок 1.32 — Просмотр параметров новой виртуальной сети

Сохранение состояния ВМ

Сохранить текущее состояние ВМ можно двумя способами:

- из главного окна менеджера ВМ;
- из консоли ВМ.

Для сохранения состояния ВМ из главного окна менеджера выберите необходимую ВМ, нажмите ПКМ по ее имени. В контекстном меню выберите «Выключить» — «Сохранить».

Для сохранения состояния ВМ из консоли ВМ в пункте меню «Виртуальная машина» выберите «Выключить» — «Сохранить». После этого работа ВМ будет приостановлена и запустится процесс сохранения состояния.

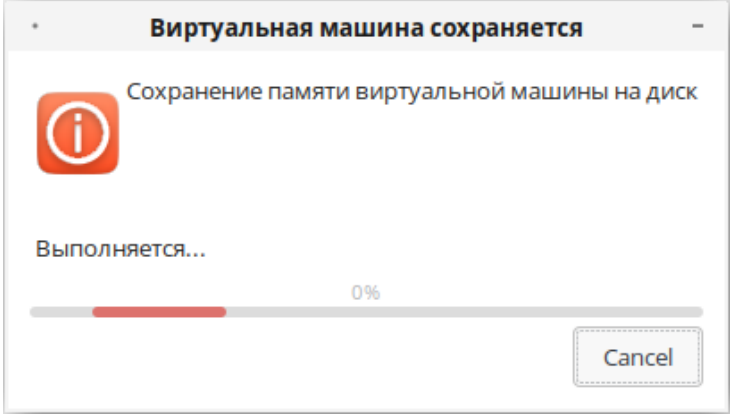

Рисунок 1.33 — Сохранение состояния ВМ

Сохраненные состояния ВМ доступны в каталоге `/var/lib/libvirt/qemu/save`.

Восстановление состояния ВМ

Восстановить ранее сохраненное состояние ВМ можно двумя способами:

- из главного окна менеджера ВМ;
- из консоли ВМ.

Для восстановления состояния ВМ из главного окна менеджера выберите необходимую ВМ, нажмите ПКМ по ее имени. В контекстном меню выберите «Восстановить».

Для восстановления состояния ВМ из консоли ВМ в пункте меню «Виртуальная машина» выберите «Восстановить».

После этого запустится процесс восстановления состояния и ВМ будет запущена.

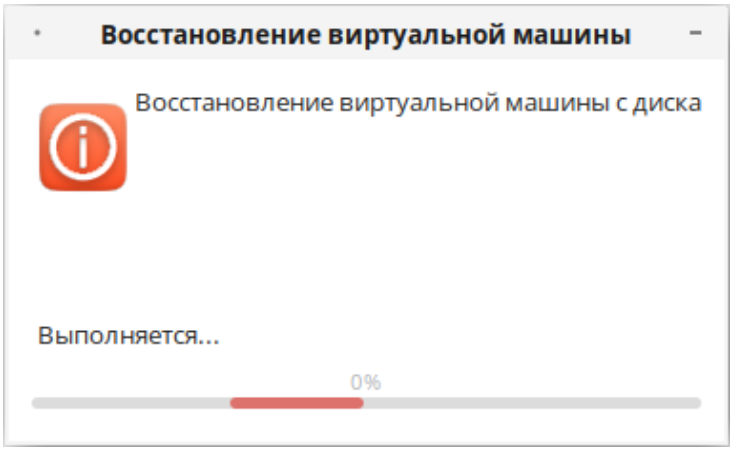

Рисунок 1.34 — Восстановление состояния ВМ

Работа со снимками ВМ

Для создания снимка (снапшота) ВМ необходимо открыть консоль ВМ и перейти на вкладку «Снимки», нажав кнопку «Управление снимками» ( ).

Для создания нового снимка ВМ нажмите кнопку «Создать новый снимок».

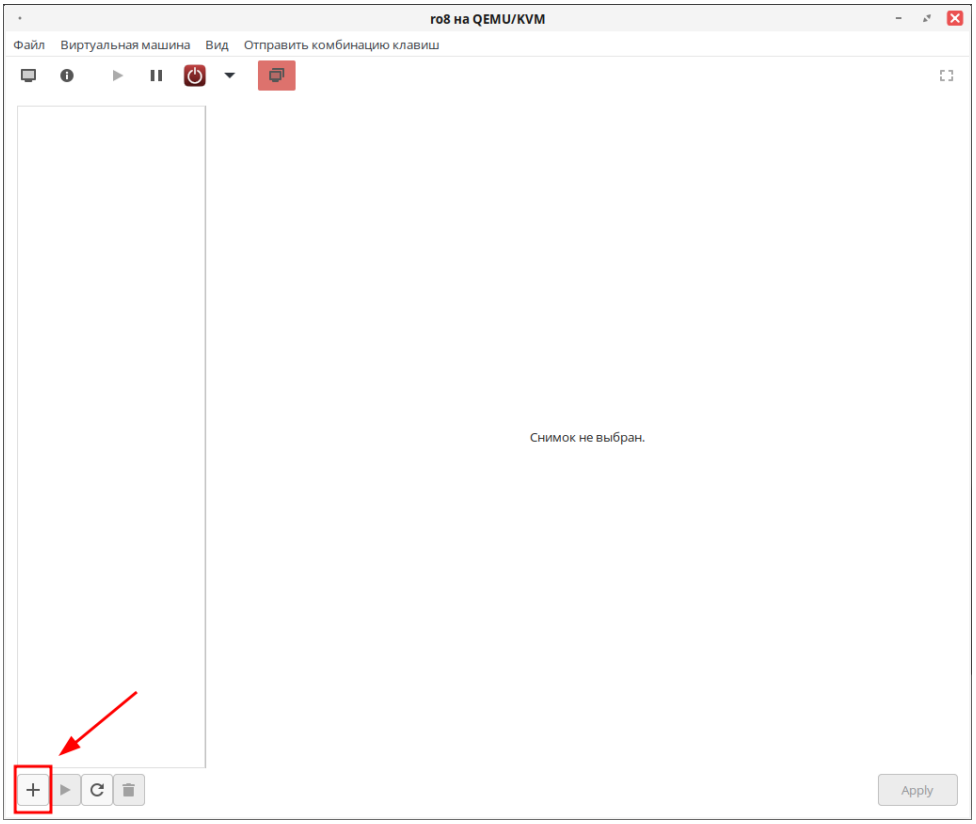

Рисунок 1.35 — Создание нового снимка ВМ

Будет открыто окно «Создание снимка», в котором необходимо указать название (обязательно) и описание (по желанию) снимка.

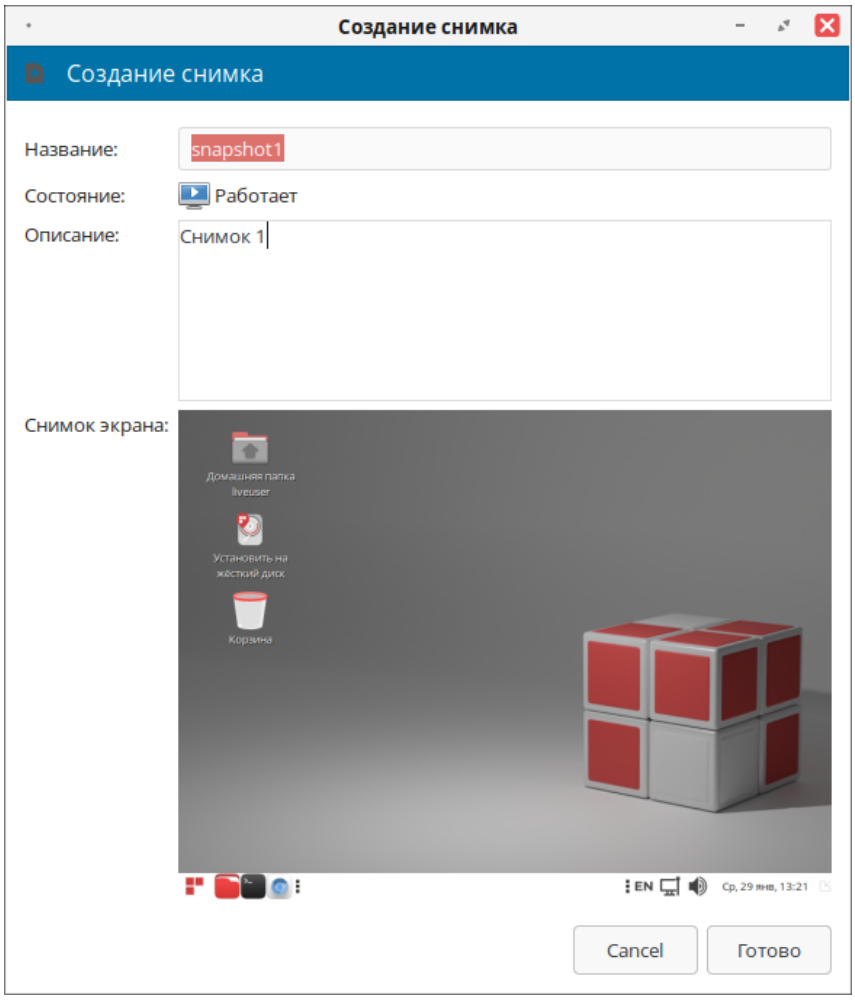

Рисунок 1.36 — Настройка параметров нового снимка ВМ

Нажмите кнопку «Готово» для запуска процесса создания снимка ВМ.

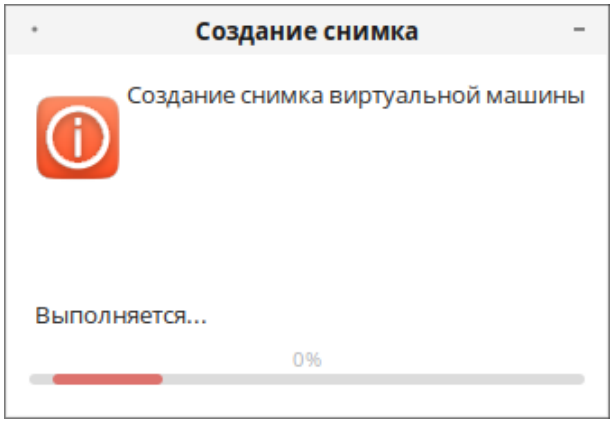

Рисунок 1.37 — Процесс создания нового снимка ВМ

После этого снимок будет доступен в списке для дальнейшей работы. При выборе конкретного снимка будет отображена подробная информация о нем.

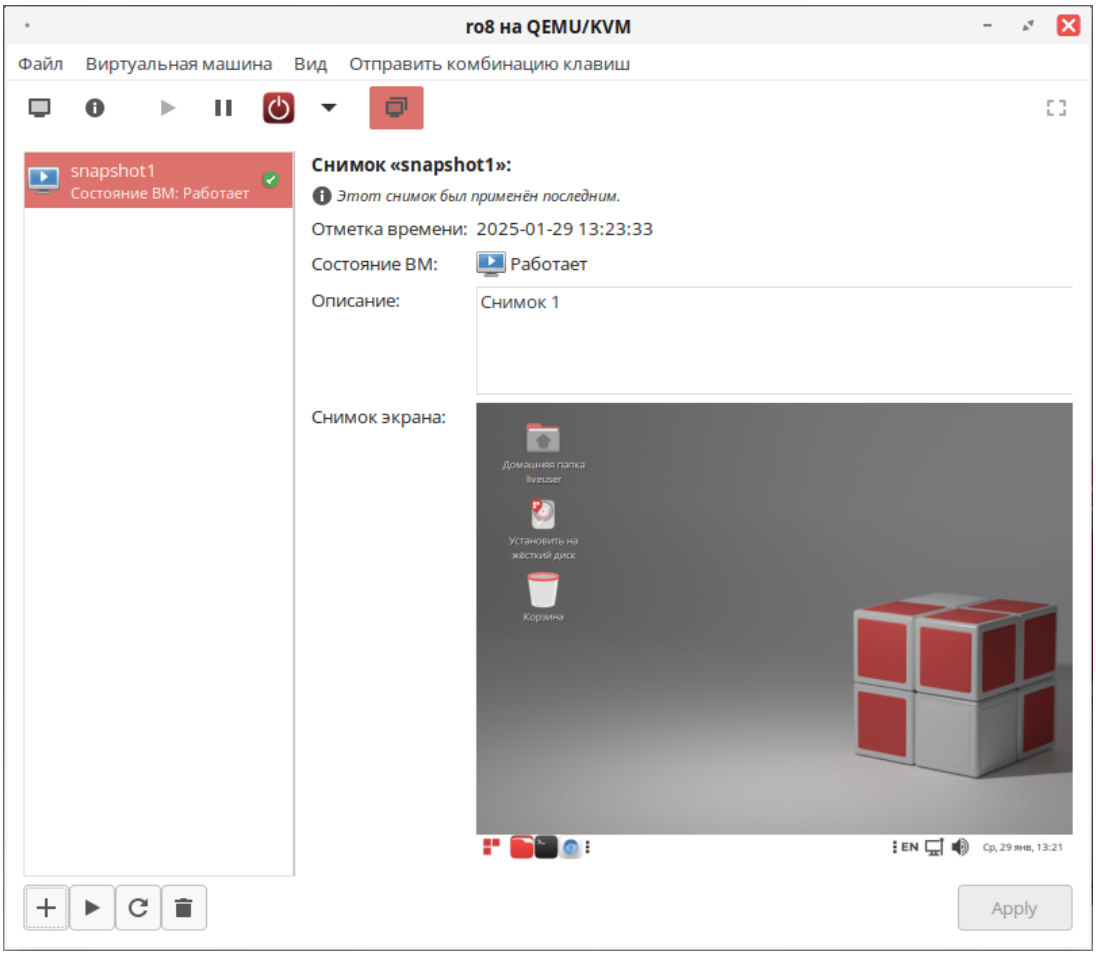

Рисунок 1.38 — Просмотр информации о снимке ВМ

Снимки можно восстанавливать и удалять, нажав ПКМ по нужному снимку либо выбрав снимок и нажав одну из кнопок — «Запустить выбранный снимок» ( ) или «Удалить выбранный снимок» ( ).

Обновить список доступных снимков можно, нажав кнопку «Обновить список снимков» ( ).

Удаление ВМ

Для удаления ВМ в менеджере необходимо выбрать нужную ВМ и нажать ПКМ по ее имени. В открывшемся контекстном меню выберите «Удалить».

Далее потребуется указать необходимость удаления пространства хранения данных. После определения всех настроек нажмите «Удалить».

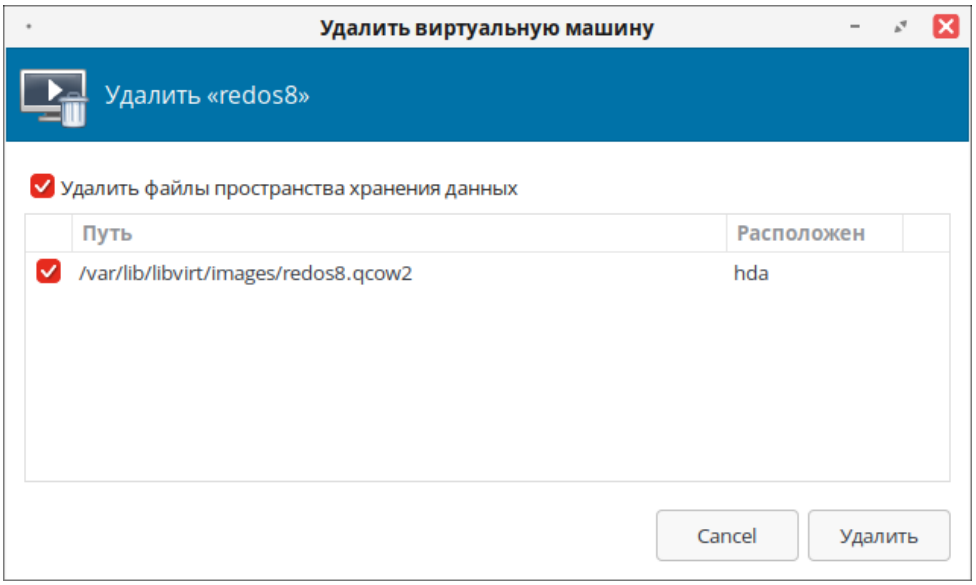

Рисунок 1.39 — Удаление ВМ

Подтвердите удаление ВМ и всех ее данных.

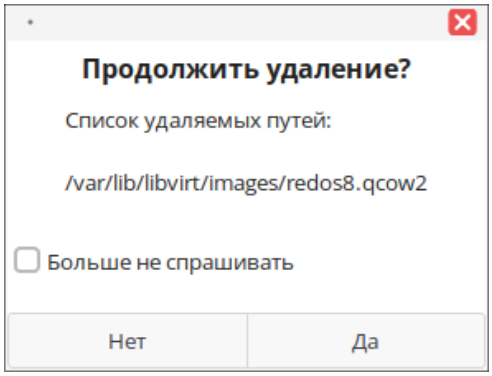

Рисунок 1.40 — Подтверждение удаления ВМ

После этого выбранная ВМ будет полностью удалена.

1.8 Безопасность в среде виртуализации

1.8.1 Доверенная загрузка ВМ

Средство виртуализации обеспечивает возможность доверенной загрузки ВМ. При выявлении нарушения целостности конфигурации виртуального оборудования ВМ средство виртуализации блокирует запуск данной ВМ.

Также средство виртуализации обеспечивает блокировку запуска ВМ при выявлении нарушения целостности файлов виртуальной базовой системы ввода-вывода (первичного загрузчика виртуальной машины) и исполняемых файлов гостевой операционной системы.

При попытке запуска ВМ с нарушением целостности конфигурации будет выведено сообщение об ошибке следующего вида:

```
ошибка: Failed to start domain 'redos8'
ошибка: доступ запрещён: доступ запрещён «QEMU»
```

1.8.2 Контроль целостности ВМ

Контроль целостности ВМ средства виртуализации обеспечивается посредством компонентов `IMA` и `afick` хостовой ОС.

Целостность ВМ обеспечивается как в процессе загрузки, так и динамически в процессе функционирования средства виртуализации. При нарушении целостности происходит немедленное информирование администратора безопасности средства виртуализации.

При попытке запуска ВМ, целостность которой была нарушена, будет выведено сообщение вида:

```
ошибка: ошибка при получении данных домена «redos8»
```

При этом в системном журнале будет зафиксировано уведомление для администратора безопасности примерно следующего вида:

```
Dec 24 17:28:12 localhost libvirtd[41711]: id=8f0d6547-cd2d-4148-
a285-e871024f3274, namespace=checksum, vm=ro8, result=deny
```

Также средством виртуализации фиксируются попытки нарушения целостности журналов `/var/log/libvirt/libvirt.log` и `/var/log/audit/audit.log`.

Записи в журналах выглядят примерно следующим образом:

```
2024-12-27 10:27:04,425 CRITICAL id=187188a8-3b7e-4508-9d66-
81d59ede036b, namespace=journal-checksum, result=WRONG
```

1.8.3 Ограничение программной среды

Для осуществления контроля за запуском компонентов программного обеспечения используется инструмент архитектуры измерения целостности хостовой ОС — `IMA`. Данный инструмент позволяет выявлять и блокировать запуск компонентов программного обеспечения, не включенных в перечень (список) компонентов, разрешенных для запуска.

Также инструмент `IMA` обеспечивает блокировку запуска компонентов программного обеспечения, целостность которого нарушена, а также компонентов, не прошедших аутентификацию с использованием свидетельств подлинности модулей.

Подробную информацию о работе и настройке инструмента `IMA` см. в разделе «Контроль целостности запускаемых компонентов» Руководства администратора. Часть 1.

1.8.4 Резервное копирование

Средство виртуализации обеспечивает резервное копирование образов ВМ и конфигураций виртуального оборудования ВМ. Также поддерживается резервное копирование параметров настройки средства виртуализации и сведений о событиях безопасности.

Для резервного копирования в средстве виртуализации используются следующие утилиты:

- `libvirt-backup-settings` — для создания резервных копий файлов конфигурации средства виртуализации;
- `libvirt-backup-vm` — для создания резервных копий ВМ и их файлов.

Синтаксис утилиты `libvirt-backup-settings` имеет следующий вид:

```
libvirt-backup-settings <каталог_для_РК>
```

В качестве обязательного аргумента утилите передается путь к каталогу, в котором будут храниться резервные копии параметров настроек средства виртуализации. В процессе создания резервной копии будет сформирован архив вида `libvirt-settings-<timestamp>.tar.gz`, в нем будут храниться все файлы настроек средства виртуализации.

Синтаксис утилиты `libvirt-backup-vm` имеет следующий вид:

```
libvirt-backup-vm <имя_ВМ> <каталог_для_РК>
```

Для создания резервной копии утилите необходимо передать два обязательных аргумента — имя ВМ и путь к каталогу хранения резервных копий. В процессе создания резервной копии так же будет сформирован архив вида `vm-<имя_ВМ>-<timestamp>.tar.gz`.

В среде виртуализации также поддерживается резервное копирование журналов безопасности. Для формирования архива с отметкой времени создания резервной копии используется команда вида:

```
tar -czf <каталог_для_РК>/libvirt-sec-journals-$(date +%Y-%m-%d_
%H-%M-%S).tar.gz /var/log/libvirt/libvirt.log* /var/log/pam.ext.log*
/var/log/audit/audit.log*
```

Создание расписания резервного копирования

Для создания расписания резервного копирования необходимо воспользоваться настройкой службы `cron` хостовой ОС. Для редактирования файла необходимо с правами пользователя `root` выполнить команду:

```
crontab -e
```

Каждая строка в системной конфигурации `cron`, расположенной в каталоге `/etc`, состоит из шести полей и команды, необходимой для выполнения:

```
<минута> <час> <число> <месяц> <день_недели> <пользователь> <ко-
манда>
```

Для резервного копирования настроек средства виртуализации в файл `cron` необходимо добавить скрипт:

```
/usr/sbin/libvirt-backup-settings <каталог_для_РК>
```

Для резервного копирования ВМ в файл `cron` необходимо добавить скрипт:

```
/usr/sbin/libvirt-backup-vm <имя_ВМ> <каталог_для_РК>
```

Для резервного копирования журналов можно использовать следующую команду:

```
tar -czf <каталог_для_РК>/libvirt-sec-journals-$(date +%Y-%m-%d_
%H-%M-%S).tar.gz /var/log/libvirt/libvirt.log* /var/log/pam.ext.log*
/var/log/audit/audit.log*
```

После этого в указанном каталоге `<каталог_для_РК>` будут созданы архивы с резервными копиями формата:

- для настроек средства виртуализации — `libvirt-settings-<timestamp>.tar.gz`;
- для виртуальных машин — `vm-<имя_ВМ>-<timestamp>.tar.gz`;
- для журналов безопасности — `libvirt-sec-journals-<timestamp>.tar.gz`.

1.8.5 Защита памяти

В средстве виртуализации обеспечивается очистка остаточной информации из памяти хоста при ее освобождении (распределении), а также удаление объектов файловой системы, используемых средством виртуализации.

Очистка памяти производится с помощью утилиты `virsh`. Уничтожение объектов файловой системы производится путем многократного перезаписывания стираемых объектов.

Для очищения только содержимого томов хранения ВМ используется команда `virsh vol-wipe`. Она позволяет безопасно уничтожить данные, ранее хранившиеся на томе без возможности чтения их в будущем. Можно использовать различные алгоритмы удаления, применив аргумент `--algorithm`. Аргумент может принимать следующие значения:

- `zero` — 1 проход записи нулями;
- `nnsa` — 3 прохода алгоритмом NNSA NAP-14.1-C;
- `dod` — 3 прохода алгоритмом DoD 5220.22-M;
- `bsi` — 9 проходов алгоритмом BSI;
- `random` — 1 проход случайной последовательностью;
- `random2` — 2 прохода случайной последовательностью;
- `schneier` — 7 проходов алгоритмом Брюса Шнайера;
- `pfitzner7` — 7 случайных проходов алгоритмом Роя Пфитцнера;
- `pfitzner33` — 33 случайных прохода алгоритмом Роя Пфитцнера;
- `gutmann` — 35 проходов алгоритмом Гутмана.

Алгоритмы становятся доступны к использованию после установки утилиты `scrub` на хостовой ОС. Утилита перезаписывает указанный файл (хранилище) случайными данными, чтобы скрыть его содержимое, и, в случае необходимости, удаляет его.

Для установки утилиты на хостовой ОС выполните команду:

```
dnf install scrub
```

После успешной установки можно перейти к очистке томов хранения ВМ. Для этого выполните команду вида:

```
virsh vol-wipe <том> [<пул_хранения>] [--algorithm <метод>]
```

Если метод очистки тома не задан, по умолчанию будет использоваться метод `nnsa`.

Пример использования утилиты:

```
vol-wipe /var/lib/libvirt/images/redos.qcow2 --algorithm bsi
```

```
Том /var/lib/libvirt/images/redos.qcow2 очищен
```

Для полного удаления как содержимого томов хранения, так и самих ВМ используется команда `virsh undefine` с дополнительными параметрами `--wipe-storage` и `--remove-all-storage`. Команда позволяет полностью удалить всю информацию о ВМ — тома хранения, конфигурационные файлы — без возможности восстановления.

Для полного удаления ВМ выполните команду вида:

```
virsh undefine <домен> --wipe-storage --remove-all-storage
```

Пример использования команды:

```
virsh undefine redos --wipe-storage --remove-all-storage
```

```
Domain 'redos' has been undefined
Освобождение тома «vda» (/var/lib/libvirt/images/redos.qcow2)...
Завершено.
Том «vda» (/var/lib/libvirt/images/redos.qcow2) был удалён.
Освобождение тома «vdb» (/var/lib/libvirt/images/redos-1.qcow2)...
Завершено.
Том «vdb» (/var/lib/libvirt/images/redos-1.qcow2) был удалён.
```

Все привязанные к домену тома будут очищены и удалены.

1.8.6 Управление потоками информации

Сетевые фильтры позволяют администраторам виртуализированных систем настраивать и применять правила фильтрации сетевого трафика на ВМ, а также управлять его параметрами.

Правила сетевых фильтров применяются на хостовой ОС при запуске ВМ и могут быть изменены в процессе ее работы. Несколько ВМ могут использовать один общий сетевой фильтр. При редактировании конфигурации такого фильтра обновляются и правила фильтрации сетевого трафика всех запущенных ВМ, использующих данный фильтр.

Подсистема фильтрации сетевого трафика позволяет настроить правила фильтрации на отдельных сетевых интерфейсах, которые настроены на определенные типы сетевых конфигураций. Поддерживаются следующие типы сетей:

- `network`;
- `ethernet` — должен использоваться в режиме `bridge`;

- bridge.

Сетевые фильтры записываются в формате XML и могут содержать ссылки на другие фильтры, правила фильтрации трафика либо их комбинацию.

Правила фильтрации могут быть организованы в связанные цепочки фильтров. Их можно представить в виде древовидной структуры, в которой отдельные цепочки фильтров будут являться «ветвями» этого дерева.

Пакеты проверяются фильтром в корневой цепочке, а затем могут быть перемещены и обработаны в других (отдельных) цепочках, после чего пакеты либо возвращаются обратно в корневую цепочку, либо принимаются (или отклоняются) правилами фильтрации в отдельной цепочке.

Система сетевой фильтрации средства виртуализации автоматически создает отдельные корневые цепочки для каждого сетевого интерфейса виртуальной машины, на котором пользователь активирует фильтрацию трафика. Пользователь может прописать правила фильтрации, которые непосредственно включаются в корневую цепочку, либо создавать цепочки фильтрации, специфичные для конкретного протокола.

Существуют следующие цепочки фильтрации трафика:

- root;
- mac;
- stp (spanning tree protocol);
- vlan (802.1Q);
- arp, rarp;
- ipv4;
- ipv6.

У каждой цепочки фильтров есть свой приоритет выполнения. В таблице 1.1 приведены приоритеты, установленные по умолчанию.

Таблица 1.1 — Цепочки фильтров и их приоритеты.

Цепочка (префикс)	Приоритет
stp	-810
mac	-800
vlan	-750
ipv4	-700
ipv6	-600
arp	-500
rarp	-400

Доступ к префиксам с более низким значением приоритета осуществляется раньше, чем к префиксам с более высоким значением.

Переменные фильтров

Для использования подсистемой фильтрации сетевого трафика зарезервированы два имени переменных: MAC и IP.

MAC — это MAC-адрес сетевого интерфейса. Правило фильтрации, которое ссылается

на эту переменную, будет автоматически создано с MAC-адресом интерфейса.

IP представляет собой IP-адрес, который гостевая ОС должна использовать на данном интерфейсе.

Элементы фильтров

Главный элемент, необходимый для всех сетевых фильтров, называется filter и имеет два возможных атрибута.

Атрибут name предоставляет уникальное имя данного фильтра.

Атрибут chain является необязательным, но позволяет лучше организовать определенные фильтры для более эффективной обработки подсистемой брандмауэра основного хоста.

В настоящее время система поддерживает только цепочки root, ipv4, ipv6, arp и rarp.

Ссылки на другие фильтры

Любой фильтр может содержать ссылки на другие фильтры. На отдельные фильтры можно ссылаться несколько раз в дереве фильтров, но ссылки между фильтрами не должны создавать циклов.

Например:

```
<filter name='clean-traffic'>
  <uuid>6ef53069-ba34-94a0-d33d-17751b9b8cb1</uuid>
  <filterref filter='no-mac-spoofing'/>
  <filterref filter='no-ip-spoofing'/>
  <filterref filter='allow-incoming-ipv4'/>
  <filterref filter='no-arp-spoofing'/>
  <filterref filter='no-other-l2-traffic'/>
  <filterref filter='qemu-announce-self'/>
</filter>
```

В данной конфигурации сетевой фильтр clean-traffic ссылается на несколько других фильтров.

Чтобы сослаться на другой фильтр, параметр filterref должен быть указан в значении filter. Данное значение должно содержать имя фильтра, на который нужно сослаться.

Новые сетевые фильтры могут быть определены в любое время и могут содержать ссылки на новые сетевые фильтры. Однако после запуска виртуальной машины все сетевые фильтры в дереве фильтров должны быть доступны. В противном случае виртуальная машина не запустится.

Правила фильтрации

Правило фильтрации всегда начинается с параметра rule, который может содержать до трех атрибутов:

- action — обязательный атрибут, может принимать следующие значения:
  - drop — отклоняет пакет без дальнейшего анализа;
  - reject — генерирует сообщение об отказе без дальнейшего анализа;
  - accept — принимает пакет без дальнейшего анализа;
  - return — проходит фильтр, при этом возвращая управление вызывающему фильтру для дальнейшего анализа;
  - continue — переходит к следующему правилу для дальнейшего анализа.

- `direction` — обязательный параметр, может принимать следующие значения:
  - `in` — если правило предназначено для входящего трафика;
  - `out` — если правило предназначено для исходящего трафика;
  - `inout` — если правило предназначено для входящего и исходящего трафика.
- `priority` — необязательный параметр, задает приоритет правила и контролирует порядок, в котором правило будет выполняться относительно других;
- `statematch` — необязательный параметр, возможные значения - `0` или `false` — для отключения проверки состояния базового соединения; по умолчанию `true`.

Пример использования правил фильтрации:

```
<filter name='no-ip-spoofing' chain='ipv4'>
  <uuid>fce8ae33-e69e-83bf-262e-30786c1f8072</uuid>
  <rule action='drop' direction='out' priority='500' >
    <ip match='no' srcipaddr='$IP'/>
  </rule>
</filter>
```

Пример использования сетевого фильтра

Для настройки сетевых фильтров на хостовой ОС необходимо определить `MAC`-адрес шлюза командой:

```
ifconfig
```

```
...
virbr0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
  inet 192.168.122.1 netmask 255.255.255.0 broadcast 192.168.122.255
  ether 52:54:00:d8:21:b1 txqueuelen 1000 (Ethernet)
  RX packets 21402 bytes 1187793 (1.1 MiB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 35100 bytes 77208945 (73.6 MiB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Затем необходимо разрешить редактирование `XML`-конфигураций ВМ. Для этого в менеджере ВМ требуется выбрать пункт меню «Правка» - «Параметры» и на вкладке «Общие» установить флаг для параметра `Enable XML editing`.

После этого в консоли ВМ необходимо открыть подробные сведения об оборудовании. Далее следует открыть вкладку настроек сети `NIC`, перейти в настройки `XML` и назначить фильтр на сетевой интерфейс.

Для этого в конфигурации вручную должны быть прописаны следующие строки:

```
<filterref filter="clean-traffic-gateway">
   <parameter name="GATEWAY_MAC" value="52:54:00:d8:21:b1"/>
</filterref>
```

где `GATEWAY_MAC` — это `MAC`-адрес шлюза.

Примечание. Внесенные в `XML`-конфигурацию изменения вступят в силу **после выключения** ВМ.

Итоговая настройка будет выглядеть примерно следующим образом:

Подробности XML

```
<interface type="network">
  <mac address="52:54:00:4f:ec:8d"/>
  <source network="default"/>
  <model type="virtio"/>
  <filterref filter="clean-traffic-gateway">
    <parameter name="GATEWAY_MAC" value="52:54:00:d8:21:b1"/>
  </filterref>
  <address type="pci" domain="0x0000" bus="0x01" slot="0x00" function="0x0"/>
</interface>
```

Рисунок 1.41 — Настройка сетевого фильтра

Описание правила `clean-traffic` можно просмотреть командой:

```
virsh nwfilter-dumpxml clean-traffic
```

```
<filter name='clean-traffic' chain='root'>
  <uuid>3a4de05c-e2e4-4e7c-952f-741160051ad0</uuid>
  <filterref filter='no-mac-spoofing'/>
  <filterref filter='no-ip-spoofing'/>
  <rule action='accept' direction='out' priority='-650'>
    <mac protocolid='ipv4'/>
  </rule>
  <filterref filter='allow-incoming-ipv4'/>
  <filterref filter='no-arp-spoofing'/>
  <rule action='accept' direction='inout' priority='-500'>
    <mac protocolid='arp'/>
  </rule>
  <filterref filter='no-other-l2-traffic'/>
  <filterref filter='qemu-announce-self'/>
</filter>
```

1.8.7 Регистрация событий безопасности

Средство виртуализации обеспечивает регистрацию событий безопасности и осуществляет сбор и хранение записей в журналах событий безопасности.

К событиям безопасности относятся следующие события:

- успешные и не успешные попытки аутентификации пользователей средства виртуализации;
- доступ пользователей средства виртуализации к виртуальным машинам;
- создание и удаление виртуальных машин;
- запуск и остановка средства виртуализации с указанием причины остановки;
- запуск и остановка виртуальных машин с указанием причины остановки;
- изменение ролевой модели;
- изменение конфигурации средства виртуализации;
- изменение конфигураций виртуальных машин;

- факты нарушения целостности объектов контроля.

К журналам событий безопасности относятся следующие файлы:
- `/var/log/libvirt/libvirt.log` – регистрация действий в средстве виртуализации;
- `/var/log/pam.ext.log` – регистрация успешных и не успешных попыток аутентификации пользователей средства виртуализации;
- `/var/log/audit/audit.log` – регистрация событий аудита, таких как запуск/остановка сервисов, предоставление прав, доступ к объектам контроля и пр.

Журналы безопасности доступны только для чтения и только пользователю с правами администратора безопасности средства виртуализации (`security-admin`).

Для чтения журналов безопасности используется утилита `libvirt-journal-read`. Синтаксис утилиты имеет следующий вид:

```
libvirt-journal-read <main | pam | audit>
```

В качестве аргументов утилита принимает один из следующих параметров:
- `main` — выводит содержимое журнала `/var/log/libvirt/libvirt.log`;
- `pam` — выводит содержимое журнала `/var/log/pam.ext.log`;
- `audit` — выводит содержимое журнала `/var/log/audit/audit.log`.

Записи в журналах событий безопасности имеют примерно следующий вид:
- записи журнала `/var/log/libvirt/libvirt.log`:

```
2025-01-06 10:01:23,669 INFO id=d9e9fbda-f326-4cec-8ed9-a6088ea-
22806, namespace=RBAC, permission=org.libvirt.rbac, (user=virt-sup,
session_id=16), result=allow
2025-01-06 10:01:38,699 INFO id=86762d9a-5c30-45d4-8ad2-f9e41f8f-
9c5c, namespace=RBAC, permission=org.libvirt.rbac, (user=virt-sup,
session_id=16), result=allow
2025-01-06 10:01:38,834 WARNING id=d07d04ec-5f67-4def-a3b5-2a31c-
f615e4b, namespace=RoleModel, user=virt-sup, role=security-admin,
action=revoke-role
2025-01-06 10:01:44,215 INFO id=9eca307e-a44b-48e8-89ea-b471ceaa-
c173, namespace=RBAC, permission=org.libvirt.journal, (user=virt-sup,
session_id=16), result=deny
```

- записи журнала `/var/log/pam.ext.log`:

```
[Mon Jan 6 09:20:03 2024] [e4901b4a-fe6c-4ab5-ba13-2e43858b6cb0]
INFO Session opened for user gdm.
[Mon Jan 6 09:20:19 2024] [f1ac23ca-a98a-40fb-bbf1-2ac25e22528d]
INFO User sec-adm logging in.
[Mon Jan 6 09:20:19 2024] [2cccb4ed-3d28-4ad7-8925-dfd1d92dfcda]
INFO Session opened for user sec-adm.
[Mon Jan 6 09:20:19 2024] [aaa1ded6-cac7-40b0-a1da-93007185eea9]
INFO Session opened for user sec-adm.
```

- записи журнала /var/log/audit/audit.log:

```
   type=USER_END msg=audit(1680764764.307:3402937): pid=2930119
uid=0 auid=4294967295 ses=4294967295 subj=system_u:system_r:policy-
kit_t:s0 msg='op=PAM:session_close grantors=pam_keyinit,pam_limits,
pam_systemd,pam_unix acct="root" exe="/usr/bin/sudo" hostname=?
addr=? terminal=? res=success'UID="root" AUID="unset"
   type=CRED_DISP msg=audit(1680764764.307:3402938): pid=2930119
uid=0 auid=4294967295 ses=4294967295 subj=system_u:system_r:policy-
kit_t:s0 msg='op=PAM:setcred grantors=pam_env,pam_faillock,pam_unix
acct="root" exe="/usr/bin/sudo" hostname=? addr=? terminal=? res=suc-
cess'UID="root" AUID="unset"
   type=USER_START msg=audit(1680764764.328:3402939): pid=2930092
uid=1004 auid=1004 ses=16 subj=unconfined_u:unconfined_r:unconfined-
_t:s0-s0:c0.c1023 msg='op=PAM:session_open grantors=pam_keyinit,
pam_limits,pam_systemd,pam_unix acct="root" exe="/usr/bin/pkexec"
hostname=localhost.localdomain addr=? terminal=pts/5 res=success'
UID="sec-adm" AUID="sec-adm"
```

Стандартная настройка ротации и архивации журнала производится при достижении размера 20 МБ. Настройка параметров осуществляется только администратором безопасности средства виртуализации (security-admin) в файле /etc/logrotate.d/libvirt.

Содержимое файла по умолчанию имеет следующий вид:

```
/var/log/pam.ext.log
/var/log/libvirt/libvirt.log {
  missingok
  maxsize 20M
  compress
  rotate 10
  copytruncate
  delaycompress
}
```

При достижении файлом журнала максимально установленного размера будет выполнено его архивирование. Копия журнала будет сохранена в каталоге /var/log/libvirt/ с указанием даты ее создания.

Пример файла:

```
ls -l /var/log/libvirt/
```

```
-rw-------. 1 root root     0 июн 30 17:51 libvirt.log
-rw-------. 1 root root 22,2M июн 30 17:51 libvirt.log-20250630
```

Средства контейнеризации

2. Средство контейнеризации

2.1 Общие сведения

Контейнеризация — метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного. Эти экземпляры (обычно называемые контейнерами) с точки зрения выполняемых в них процессов идентичны отдельному экземпляру операционной системы. Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга.

В отличие от аппаратной виртуализации, при которой эмулируется аппаратное окружение и может быть запущен широкий спектр гостевых операционных систем, в контейнере может быть запущен экземпляр операционной системы только с тем же ядром, что и у хостовой операционной системы (все контейнеры узла используют общее ядро). При этом при контейнеризации отсутствуют дополнительные ресурсные накладные расходы на эмуляцию виртуального оборудования и запуск полноценного экземпляра операционной системы, характерные при аппаратной виртуализации.

В РЕД ОС в качестве средства контейнеризации используется контейнерная платформа Docker.

Docker — программное обеспечение для автоматизации развертывания и управления приложениями в средах с поддержкой контейнеризации. Позволяет "упаковать" приложение со всем его окружением и зависимостями в контейнер, который может быть развернут на любой Linux-подобной системе с поддержкой контрольных групп в ядре, а также предоставляет набор команд для управления этими контейнерами.

Для экономии пространства хранения проект использует файловую систему `aufs` с поддержкой технологии каскадно-объединенного монтирования: контейнеры используют образ базовой операционной системы, а изменения записываются в отдельную область. Также поддерживается размещение контейнеров в файловой системе `btrfs` с включенным режимом копирования при записи.

В состав программных средств входит демон — сервер контейнеров, клиентские средства, позволяющие из интерфейса командной строки управлять образами и контейнерами, а также REST API, позволяющий управлять контейнерами программно. Демон обеспечивает полную изоляцию запускаемых на узле контейнеров на уровне файловой системы (у каждого контейнера собственная корневая файловая система), на уровне процессов (процессы имеют доступ только к собственной файловой системе контейнера, а ресурсы разделены средствами `containerd`), на уровне сети (каждый контейнер имеет доступ только к привязанному к нему сетевому пространству имен и соответствующим виртуальным сетевым интерфейсам).

Набор клиентских средств позволяет запускать процессы в новых контейнерах,

останавливать и запускать контейнеры, приостанавливать и возобновлять процессы в контейнерах. Набор команд позволяет осуществлять мониторинг запущенных процессов. Новые образы возможно создавать из специального файла, также возможно записать все изменения, произведенные в контейнере, в новый образ. Все команды могут работать как с `docker`-сервисом локальной системы, так и с любым сервером Docker, доступным по сети.

Кроме того, в интерфейсе командной строки встроены возможности по взаимодействию с различными централизованными хранилищами образов, в которых размещены предварительно собранные образы контейнеров.

2.2 Установка среды контейнеризации

Существует два варианта установки среды контейнеризации:

- установка в процессе развертывания новой системы РЕД ОС;
- установка в существующей системе РЕД ОС.

2.2.1 Установка в процессе развертывания новой системы РЕД ОС

После запуска интерактивной установки РЕД ОС с установочного носителя (`USB`, `CD-ROM`, `PXE`) параметры будущей системы настраиваются в штатном режиме (выбор языка, часового пояса, раскладки клавиатуры, источника установки, настройка сети), кроме параметра «Выбор программ».

Для развертывания среды контейнеризации необходимо выполнить установку РЕД ОС конфигурации Сервер (с графическим интерфейсом или минимальный).

При выборе конфигурации Сервер (с графическим интерфейсом или минимальный) следует определить дополнительное программное обеспечение — *Средства контейнеризации.*

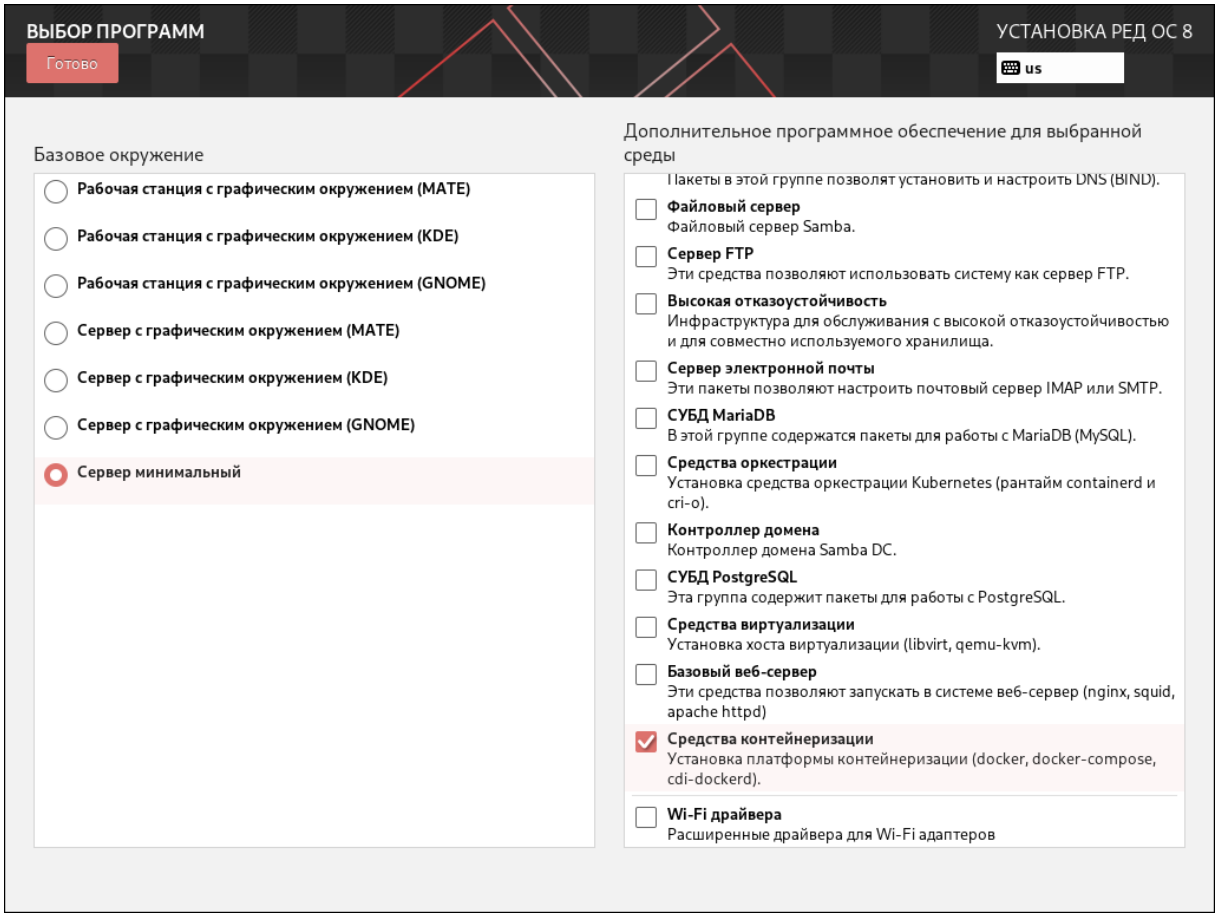

Рисунок 2.1 — Установка средства контейнеризации

После определения всех настроек нажмите кнопку «Начать установку».

Для завершения процесса установки потребуется перезагрузить систему и принять Лицензионное соглашение.

После авторизации в системе необходимо проверить статус сервиса `docker` командой:

```
systemctl status docker
```

В статусе должно быть указано `active (running)`. Если по каким-либо причинам статус не активен, выполните команду запуска сервиса с правами пользователя `root`:

```
systemctl start docker
```

После этого ОС будет готова к использованию.

2.2.2 Установка в существующей системе РЕД ОС

Для развертывания среды контейнеризации необходимо выполнить команду:

```
dnf install docker-ce docker-ce-cli
```

После успешной установки необходимо запустить сервис контейнеризации `docker` и добавить его в автозагрузку:

```
systemctl enable docker --now
```

Убедитесь, что сервис запущен, проверив статус запущенной службы:

```
systemctl status docker
```

В статусе должно быть отображено `active (running)`.

Для получения информации об установленном `docker` выполните команду:

```
docker info
```

При корректной настройке будет получен соответствующий ответ от сервиса Docker.

2.3 Доступ к среде контейнеризации

По умолчанию доступ к среде контейнеризации и запуску сервисов имеет только суперпользователь. Демон `docker` подключается к сокету Unix, к которому также имеет доступ только суперпользователь.

Для использования и управления средой контейнеризации обычным пользователем необходимо добавить его в отдельную группу, пользователям которой будет разрешено выполнять необходимые манипуляции. Такая группа создается в процессе установки среды контейнеризации и имеет название `docker`.

Добавьте необходимого пользователя в группу `docker` командой:

```
usermod -aG docker <имя_пользователя>
```

Для применения изменений выйдите из сеанса пользователя и авторизуйтесь снова либо перезагрузите систему. Проверить назначенные права можно, выполнив загрузку тестового образа с правами обычного пользователя:

```
docker info
```

При корректной настройке будет получен соответствующий ответ от сервиса Docker.

2.4 Компоненты средства контейнеризации

Docker использует клиент-серверную архитектуру. Клиент взаимодействует с демоном `Docker`, который выполняет задачи по созданию, запуску и распространению контейнеров Docker. Клиент и демон могут работать в одной системе, также можно подключить клиент к удаленному демону `Docker`. Клиент и демон взаимодействуют с помощью REST API, через сокеты UNIX или сетевой интерфейс.

2.4.1 Dockerfile

Dockerfile — это простой текстовый файл, включающий в себя инструкции по созданию контейнера Docker. Dockerfile описывает будущую операционную систему, которая будет лежать в основе контейнера, а также языки, переменные среды, расположение файлов, сетевые порты и другие необходимые компоненты и действия контейнера после его запуска.

> **Важно!** Не используйте корень файловой системы хостовой ОС (/) в качестве пути для создания образа Docker — это приведет к передаче демону всего содержимого жесткого диска вашей машины. ■

Инструкции выполняются последовательно по принципу «одна строка — одна команда». Результаты наиболее важных инструкций фиксируются в виде отдельных слоев образа.

> **Примечание.** Каждая новая инструкция выполняется независимо от других — это означает, что выполнение `RUN cd /tmp` не будет иметь никакого эффекта для последующих инструкций.

Слои в образе создают только инструкции `FROM`, `RUN`, `COPY`, и `ADD`.

Формат файла имеет следующий вид:

```
<ИНСТРУКЦИЯ> <аргументы>
```

Инструкции не чувствительны к регистру, однако используется единый стиль их оформления с использованием заглавных букв.

Список инструкций, используемых в Dockerfile:

- `FROM` — определяет базовый (родительский) образ;
- `LABEL` — описывает метаданные, например, сведения о том, кто создал и поддерживает образ;
- `ENV` — устанавливает постоянные переменные среды;
- `RUN` — выполняет команду и создаёт слой образа, используется для установки в контейнер пакетов;
- `COPY` — копирует в контейнер файлы и папки;
- `ADD` — копирует файлы и папки в контейнер, может распаковывать локальные `.tar`-файлы;
- `CMD` — описывает команду с аргументами, которую нужно выполнить, когда контейнер будет запущен. Аргументы могут быть переопределены при запуске контейнера. В файле может присутствовать только одна инструкция `CMD`;
- `WORKDIR` — задаёт рабочую директорию для следующей инструкции;
- `ARG` — задаёт переменные для передачи Docker во время сборки образа;
- `ENTRYPOINT` — предоставляет команду с аргументами для вызова во время выполнения контейнера. Аргументы не переопределяются;
- `EXPOSE` — указывает на необходимость открыть порт;
- `VOLUME` — создаёт точку монтирования для работы с постоянным хранилищем.

С подробной информацией об использовании и применении Dockerfile можно ознакомиться в справочной странице:

```
man dockerfile
```

Инструкция FROM

Dockerfile должен начинаться с инструкции `FROM`.

Инструкция `FROM` задает базовый (родительский) образ, на основе которого будет создаваться новый образ. Перед инструкцией `FROM` могут быть указаны только директивы синтаксического анализатора (parser directives), комментарии или глобальные аргументы (`ARG`).

Строки, начинающиеся с символа # воспринимаются как комментарии и не обрабатываются.

Если вы хотите создать образ, основанный на Ubuntu, вы можете использовать следующую инструкцию `FROM`:

```
FROM ubuntu:latest
```

Эта инструкция указывает Docker, что будет использована последняя версия образа Ubuntu в качестве базового для создания нового образа.

Инструкция LABEL

Инструкция `LABEL` (метка) позволяет добавлять в образ метаданные. Она может включать в себя контактные сведения создателя образа, версию приложения, описание образа и т.д. Объявление меток не замедляет процесс сборки образа и не увеличивает его размер. Они содержат в себе полезную информацию об образе Docker, поэтому их рекомендуется включать в файл.

Пример использования инструкции `LABEL`:

```
LABEL maintainer="Ivan Ivanov <iva_iva@example.com>"
LABEL version="1.0"
LABEL description="This is a sample Docker image"
```

Метаданные могут быть полезны при управлении образами и контейнерами в больших проектах.

Инструкция ENV

Инструкция `ENV` позволяет задавать постоянные переменные среды, которые будут доступны в контейнере во время его выполнения.

Инструкция `ENV` подходит для задания констант. Если вы используете некоторое значение в Dockerfile несколько раз, например, при описании команд, выполняющихся в контейнере, и в будущем оно может измениться, имеет смысл записать данное значение в подобную константу.

Пример использования инструкции `ENV`:

```
ENV ADMIN="ivan"
```

Инструкция RUN

Инструкция `RUN` позволяет выполнять команды внутри контейнера во время создания образа.

Пример использования инструкции `RUN`:

```
RUN dnf update && dnf install -y
    package1
    package2
    package3
```

Данная инструкция устанавливает пакеты `package1`, `package2` и `package3` внутри контейнера.

Инструкция `RUN` может использоваться для установки зависимостей, настройки окружения, запуска скриптов и т.д. Она также может использоваться для создания новых слоев в образе Docker.

Важно помнить, что каждая инструкция `RUN` создает новый слой в образе Docker, поэтому необходимо объединять несколько команд в одну строку с помощью `&&` или использовать инструкцию `SHELL` для выполнения нескольких команд в одном слое.

Инструкция COPY

Инструкция `COPY` сообщает Docker о необходимости переноса файлов и папок из локального контекста сборки в текущую рабочую директорию образа. Если целевая директория не существует, инструкция её создаст.

Пример использования инструкции `COPY`:

```
COPY . ./app
```

Инструкция ADD

Инструкция `ADD` копирует файлы или директории из исходной директории на хостовой машине в контейнер. Пример использования инструкции `ADD`:

```
ADD app.py /app/
```

Эта инструкция копирует файл `app.py` из текущей директории на хостовой машине в директорию `/app/` внутри контейнера.

Инструкция `ADD` может использоваться для добавления файлов, архивов, конфигурационных файлов и в контейнер. Она также может использоваться для скачивания файлов из сети Интернет.

Важно помнить, что инструкция `ADD` создает новый слой в образе Docker, поэтому необходимо использовать ее с осторожностью и объединять несколько команд в один слой, если это возможно. Кроме того, не рекомендуется использовать инструкцию `ADD` для копирования больших файлов, так как это может привести к увеличению размера образа Docker.

Инструкция CMD

Инструкция `CMD` передает Docker команду, которую необходимо выполнить при запуске контейнера.

Пример использования инструкции `CMD`:

```
CMD ["python", "./my_script.py"]
```

Инструкция `CMD` может использоваться для определения стандартного поведения контейнера при запуске. Это может быть полезно для установки параметров по

умолчанию, например, для запуска веб-сервера или приложения.

Стоит отметить, что инструкция `CMD` может быть переопределена при запуске контейнера с помощью флага `docker run`. Например, если необходимо выполнить другой скрипт при запуске контейнера, можно использовать следующую команду:

```
docker run my-image python another_app.py
```

Данная команда переопределит команду, определенную в инструкции `CMD`, и запустит скрипт `another_app.py` вместо `app.py`.

Ограничения использования инструкции `CMD`:

- в одном файле Dockerfile может присутствовать только одна инструкция `CMD`. Если в файле есть несколько таких инструкций, система проигнорирует все кроме последней;
- инструкция `CMD` может иметь `exec`-форму. Если в инструкцию не входит упоминание исполняемого файла, тогда в файле должна присутствовать инструкция `ENTRYPOINT`. В таком случае обе инструкции должны быть представлены в формате `JSON`.
- аргументы командной строки, передаваемые `docker run`, переопределяют аргументы, предоставленные инструкции `CMD` в Dockerfile.

Инструкция WORKDIR

Инструкция `WORKDIR` позволяет изменить рабочую директорию контейнера. С этой директорией работают инструкции `COPY`, `ADD`, `RUN`, `CMD` и `ENTRYPOINT`, указанные за `WORKDIR`.

Пример использования инструкции `WORKDIR`:

```
WORKDIR /usr/src/my_app_directory
```

Ограничения использования инструкции `WORKDIR`:

- с помощью `WORKDIR` рекомендуется устанавливать абсолютные пути к папкам, не перемещаясь по файловой системе с помощью команды `cd` в Dockerfile;
- инструкция `WORKDIR` автоматически создаёт директорию в том случае, если она не существует;
- можно использовать несколько инструкций `WORKDIR`. Если таким инструкциям предоставляются относительные пути, то каждая из них меняет текущую рабочую директорию.

Инструкция ARG

Инструкция `ARG` определяет переменную, которую можно использовать во время сборки образа.

Пример использования инструкции `ARG`:

```
ARG VERSION=latest
```

Данная инструкция определяет переменную `VERSION` со значением `"latest"`. Указанную переменную можно использовать в других командах Dockerfile, например:

```
FROM ubuntu:$VERSION
```

Данная команда использует значение переменной `VERSION` для выбора версии Ubuntu для указания в качестве базового образа.

Инструкция `ARG` может использоваться для передачи аргументов во время сборки образа, например, версии приложения или настроек конфигурации. Это может быть полезно при автоматизации процесса сборки образов Docker.

Важно помнить, что переменные, определенные с помощью инструкции `ARG`, доступны только во время сборки образа и не сохраняются в контейнере. Кроме того, значения переменных могут быть переопределены при запуске сборки образа с помощью флага `--build-arg`.

Инструкция ENTRYPOINT

Инструкция `ENTRYPOINT` определяет исполняемый файл или команду, которые будут выполняться при запуске контейнера. Инструкция также может использоваться для определения стандартного поведения контейнера при запуске. Пример использования инструкции `ENTRYPOINT`:

```
ENTRYPOINT ["python", "app.py"]
```

Данная инструкция определяет исполняемый файл, который будет выполнен при запуске контейнера — запуск скрипта `app.py` на языке Python.

В отличие от инструкции `CMD`, инструкция `ENTRYPOINT` не может быть переопределена при запуске контейнера с помощью команды `docker run`. Однако, если необходимо добавить аргументы или параметры при запуске контейнера, можно использовать инструкцию `CMD` совместно с инструкцией `ENTRYPOINT`.

Пример совместного использования инструкций `ENTRYPOINT` и `CMD`:

```
ENTRYPOINT ["python", "app.py"]
CMD ["--port", "8080"]
```

Эти инструкции определяют исполняемый файл и параметры, которые будут переданы ему при запуске контейнера. В данном случае, при запуске контейнера будет запущен скрипт `app.py` на языке Python с параметром `--port`, равным `8080`. Если необходимо изменить значение параметра `--port`, можно переопределить его при запуске контейнера с помощью флага `docker run`:

```
docker run my-image --port 9000
```

Данная команда переопределит значение параметра `--port` и запустит контейнер с параметром `--port`, равным `9000`.

Инструкция EXPOSE

Инструкция `EXPOSE` указывает, какие порты планируется открыть для того, чтобы через них можно было бы связаться с работающим контейнером. Данная инструкция не открывает порты. Она является средством общения между тем, кто собирает образ, и тем, кто запускает контейнер.

Для того чтобы открыть порт (или порты) и настроить перенаправление портов, необходимо выполнить команду `docker run` с ключом `-p`. Если использовать ключ `-P`, открыты будут все порты, указанные в инструкции `EXPOSE`.

Пример использования инструкции `EXPOSE`:

```
EXPOSE 8000
```

Инструкция VOLUME

Инструкция `VOLUME` позволяет указать каталог, который контейнер будет использовать для постоянного хранения файлов и для работы с такими файлами.

Пример использования инструкции `VOLUME`:

```
VOLUME /my_volume
```

2.4.2 Docker Registry

В централизованном хранилище образов хранятся доступные образы Docker.

Общедоступным хранилищем образов является Docker Hub, по умолчанию Docker настроен на поиск образов в нем. Также есть возможность настройки своего собственного хранилища образов.

При использовании команд `docker pull` или `docker run` требуемые образы будут извлекаться из настроенного хранилища образов. Когда используется команда `docker push`, образ помещается в настроенное хранилище образов.

2.4.3 Docker Daemon

Демон Docker (`dockerd`) прослушивает запросы Docker API и управляет объектами Docker, такими как образы, контейнеры, сети и тома. Демон также может взаимодействовать с другими демонами для управления службами Docker.

Существует два способа настройки демона Docker:

- использование файла конфигурации `JSON`, является предпочтительным вариантом, так как он хранит все конфигурации в одном месте;
- использование флагов при запуске `dockerd`.

Чтобы настроить демон Docker с помощью файла `JSON`, необходимо создать файл `/etc/docker/daemon.json`.

Пример файла конфигурации `JSON`:

```
{
  "experimental": true,
  "log-level": "warn",
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "5"
  }
}
```

В данной конфигурации описывается настройка журнала. Значение `warn` параметра `log-level` сообщает, что в журнале будут отображаться записи со статусом `Warning`. Журнал будет вестись в формате `json`-файла с максимальным размером файла `10Мб`, количество журналов не должно превышать `5` (при достижении этого значения журналы будут перезаписываться).

Синтаксис ручного запуска демона:

```
dockerd <опции>
```

Пример запуска демона вручную:

```
dockerd --debug
--tls=true
--tlscert=/var/docker/server.pem
--tlskey=/var/docker/serverkey.pem
--host tcp://192.168.59.3:2376
```

В данной конфигурации демон Docker работает в режиме отладки, использует `TLS` и прослушивает трафик, направляемый на `192.168.59.3` через порт `2376`.

Демон Docker сохраняет все данные в одном каталоге, что помогает отслеживать все, что связано с Docker, включая контейнеры, образы, тома. Каталогом по умолчанию является `/var/lib/docker`.

Часто используемые параметры:

- `--config-file="/etc/docker/daemon.json"` — путь к файлу конфигурации `JSON`;
- `-D`, `--debug=true | false` — режим отладки, по умолчанию значение `false`;
- `-H`, `--host=tcp://[host:port] | unix://[/path/to/socket]` — привязка одного или нескольких сокетов;
- `--iptables=true | false` — добавление правил `iptables`, значение по умолчанию `true`;
- `--isolation="default"` — тип изоляции контейнеров, значение по умолчанию `default`.

С подробной информацией о доступных параметрах можно ознакомиться, выполнив:

```
dockerd --help
```

2.4.4 Docker Client

Клиент Docker (`docker`) — это основной способ взаимодействия многих пользователей с Docker. Когда выполняются такие задачи, как запуск контейнера, клиент отправляет команды демону `dockerd`, который их выполняет. Команда `docker` при работе обращается к Docker API. Клиент Docker может взаимодействовать с несколькими демонами.

Синтаксис использования клиента:

```
docker [<опции>] <команда> [<аргументы...>]
```

Основные команды управления описаны в разделе «Управление контейнерами». Подробнее о работе с клиентом Docker см. в справке:

```
docker --help
```

2.4.5 Объекты Docker

Docker Image

Docker Image — неизменяемые файлы (образы), из которых можно развернуть контейнер. Образы могут быть основаны на каких-либо других образах (шаблонах) с добавлением некоторых дополнительных настроек.

Перед созданием контейнера Docker имеет возможность верифицировать образ. Если образ не изменен — контейнер запускается (создается). По умолчанию данная функция отключена, для включения верификации образов необходимо в файл `daemon.json` добавить следующую строку:

```
"disable-validation": false
```

Container

Container — исполняемый экземпляр образа.

Можно создавать, запускать, останавливать, перемещать или удалять контейнеры с помощью Docker API или командной строки. Есть возможность подключить контейнер к одной или нескольким сетям, подключить к нему хранилище или создать новый образ на основе его текущего состояния.

По умолчанию контейнер хорошо изолирован от других контейнеров и своей хост-машины. Можно контролировать, насколько изолированы сеть контейнера, хранилище или другие базовые подсистемы от других контейнеров или от хост-системы.

Контейнер определяется своим образом, а также любыми параметрами конфигурации, которые предоставляются ему при создании или запуске. При удалении контейнера любые изменения его состояния, не сохраненные в постоянном хранилище, уничтожаются.

2.4.6 Docker volume

Docker volume — механизм, позволяющий сохранять данные вне контейнера Docker. Он представляет собой отдельный файл или директорию на хост-системе, которые монтируются внутрь контейнера. Это позволяет сохранять данные между запусками контейнеров и обмениваться данными между контейнерами. Docker volumes также обеспечивают удобный способ резервного копирования данных и обновления приложений без потери данных.

Синтаксис команлы имеет следующий вид:

```
docker volume <команда>
```

Docker volume позволяет управлять томами с помощью следующих команд:

- `create` — создание тома;
- `inspect` — отображение информации о томе;
- `ls` — отображение списка томов;
- `prune` — удаление неиспользуемых томов;
- `rm` — удаление одного или нескольких томов;
- `update` — обновление тома (только для кластерных томов).

Docker volume create

Команда создает новый том, который контейнеры могут использовать и хранить в нем данные. Если имя не указано, Docker генерирует случайное имя.

Синтаксис имеет следующий вид:

```
docker volume create [<опции>] [<том>]
```

Опции команды:

- `--driver`, `-d` — имя драйвера тома, значение по умолчанию `local`;

- `--opt`, `-o` — специальные параметры драйвера;
- `--label <метка>` — указать метку для тома.

Подробную информацию об использовании команды см. в справке:

```
docker volume create --help
```

Примеры использования:
1. Создание нового тома:

```
docker volume create hello
```

2. Создание тома `tmpfs` с именем `foo` размером `100 МБ` и `uid 1000`:

```
docker volume create --driver local
    --opt type=tmpfs
    --opt device=tmpfs
    --opt o=size=100m,uid=1000
    foo
```

3. Создание тома `bfrfs` с именем `foo` и монтированием раздела `/dev/sda2`:

```
docker volume create --driver local
    --opt type=btrfs
    --opt device=/dev/sda2
    foo
```

Docker volume inspect

Команда отображает информацию о томе. По умолчанию эта команда отображает все результаты в формате `JSON`.

Синтаксис команды:

```
docker volume inspect [<опции>] <том> [<том>...]
```

Доступная опция:
- `--format`, `-f` — преобразование формата вывода с использованием пользовательского шаблона `json`.

Подробную информацию об использовании команды см. в справке:

```
docker volume inspect --help
```

Примеры использования:

```
docker volume inspect new_volume
```

```
[
  {
    "CreatedAt": "2024-12-15T13:45:16Z",
    "Driver": "local",
```

```
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/8140a838303144125b4f-
54653b47ede0486282c623c3551fbc7f390cdc3e9cf5/_data",
        "Name": "new_volume",
        "Options": {},
        "Scope": "local"
      }
    ]
```

2. Форматирование выходных данных с использованием шаблона для печати свойства `Mountpoint`:

```
docker volume inspect --format ' .Mountpoint ' new_volume
```

```
/var/lib/docker/volumes/new_volume/_data
```

Docker volume ls

Команда отображает все тома, известные Docker. Результат может быть отфильтрован с помощью опции `-f` (`--filter`).

Синтаксис команды:

```
docker volume ls [<опции>]
```

Доступные опции:

- `--filter`, `-f <фильтр>` — укажите значение фильтра (например, `driver=local`);
- `--format <строка>` — преобразование формата вывода с использованием пользовательского шаблона `table`, вывод в формате таблицы с заголовками столбцов (по умолчанию);
- `--quiet`, `-q` — отображение только имен томов.

Подробную информацию об использовании команды см. в справке:

```
docker volume ls --help
```

Примеры использования:

1. Отображение всех доступных томов:

```
docker volume ls
```

```
DRIVER      VOLUME NAME
local       test_1
local       test_2
```

2. Использование фильтра для отображения доступных томов:

```
docker volume ls -f driver=local
```

```
DRIVER      VOLUME NAME
local       test_1
local       test_2
```

Docker volume prune

Команда удаляет все неиспользуемые локальные тома. Неиспользуемыми томами считаются тома, на которые не ссылаются никакие контейнеры.

Синтаксис команды:

```
docker volume prune [<опции>]
```

Доступные опции:

- `--filter <фильтр>` — укажите значение фильтра, например, `label=<label>`.
- `--force`, `-f` — не запрашивать подтверждение.

Пример использования:

```
docker volume prune
```

```
WARNING! This will remove all local volumes not used by at least
one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
07c7bdf3e34ab76d921894c2b834f073721fccfbbcba792aa7648e3a7a664c2e
my-named-vol

Total reclaimed space: 36 B
```

Docker volume rm

Команда позволяет удалить один или несколько томов. Однако не может быть удален том , используемый контейнером.

Синтаксис команды:

```
docker volume rm [<опции>] <том> [<том_2>...]
```

Доступная опция:

- `--force`, `-f` — принудительное удаление одного или нескольких томов.

Пример использования:

```
docker volume rm hello
```

2.5 Управление контейнерами

2.5.1 Docker Container

Docker Container используется для управления контейнерами.

Синтаксис команды:

```
docker container <команда>
```

Используемые команды управления контейнерами:

- `docker container attach` — прикрепить локальные стандартные потоки ввода, вывода и вывода ошибок к запущенному контейнеру;

- `docker container commit` — создание нового образа из изменений контейнера;
- `docker container cp` — копирование файлов/папок между контейнером и хостовой файловой системой;
- `docker container create` — создание нового контейнера;
- `docker container diff` — проверка изменений файлов или каталогов в файловой системе контейнера;
- `docker container exec` — выполнение команды в запущенном контейнере;
- `docker container export` — экспорт файловой системы контейнера в виде `tar`-архива;
- `docker container inspect` — вывод подробной информации об одном или нескольких контейнерах;
- `docker container kill` — принудительное завершение одного или нескольких запущенных контейнеров;
- `docker container logs` — просмотр журналов регистрации контейнера;
- `docker container ls` — вывод списка контейнеров;
- `docker container pause` — приостановка всех процессов в одном или нескольких контейнерах;
- `docker container port` — список портов контейнера;
- `docker container prune` — удаление всех остановленных контейнеров;
- `docker container rename` — изменение имени контейнера;
- `docker container restart` — перезапуск одного или нескольких контейнеров;
- `docker container rm` — удаление одного или нескольких контейнеров;
- `docker container run` — создание и запуск нового контейнера из образа;
- `docker container start` — запуск одного или нескольких остановленных контейнеров;
- `docker container stat` — отображение статистики использования ресурсов контейнера(ов) в реальном времени;
- `docker container stop` — остановка одного или нескольких запущенных контейнеров;
- `docker container top` — отображение запущенных процессов контейнера;
- `docker container unpause` — отключение всех процессов в одном или нескольких контейнерах;
- `docker container update` — обновление конфигурации одного или нескольких контейнеров;
- `docker container wait` — блокировка остановки одного или нескольких контейнеров, вывод их кодов выхода.

При использовании команд допускается сокращение записи до вида:

```
docker <команда>
```

Далее будут рассмотрены основные команды управления контейнерами.

С подробной информацией обо всех командах и их работе можно ознакомиться, выполнив:

```
docker container <команда> --help
```

2.5.2 Docker build

Команда `docker build` используется для создания образа на основе файла Dockerfile и контекста.

Контекст — это набор файлов, находящихся по пути, определенному с помощью переменной `PATH` или `URL`. `PATH` — это директория в локальной системе, а `URL` — это удаленный репозиторий. Контекст сборки обрабатывается рекурсивно, поэтому `PATH` включает как директорию, так и все ее поддиректории, а `URL` — как репозиторий, так и все его субмодули.

Синтаксис команды:

```
docker build [<опции>] PATH | URL | -
```

Опции команды:

- `-f`, `--file <string>` — имя Dockerfile;
- `--isolation <string>` — технология изоляции контейнеров, по умолчанию `default`;
- `--label <list>` — установка метаданных для образа;
- `-m`, `--memory <bytes>` — ограничение памяти;
- `--network <string>` — установить режим работы с сетью для инструкций `RUN` во время сборки (по умолчанию `default`);
- `--rm` — удалять промежуточные контейнеры после успешной сборки (по умолчанию `true`).

2.5.3 Docker image

Команда `docker image` предназначена для управления образами.

Синтаксис команды:

```
docker image <команда>
```

Список доступных команд:

- `docker image build` — создание образа из Docker-файла;
- `docker image history` — вывод истории образа;
- `docker image import` — импорт содержимого из `tarball` для создания образа файловой системы
- `docker image inspect` — вывод подробной информации об одном или нескольких образах;
- `docker image load` — загрузка образа из `tar`-архива или `STDIN`;
- `docker image ls` — вывод списка образов;
- `docker image prune` — удаление неиспользуемых образов;
- `docker image pull` — загрузка образа из `registry`;
- `docker image push` — загрузка образа в `registry`;
- `docker image rm` — удалить один или несколько образов;
- `docker image save` — сохранить один или несколько образов в `tar`-архив (по умолчанию передается в `STDOUT`);
- `docker image tag` — создать тег `TARGET_IMAGE`, который ссылается на `SOURCE_IMAGE`.

2.5.4 Docker images

Команда `docker images` используется для вывода списка доступных образов. Синтаксис команды:

```
docker images [<опции>] [<репозиторий>[:<тег>]]
```

Используемые опции команды:

- `--all`, `-a` — показать все образы (по умолчанию промежуточные образы скрыты);
- `--digests` — показать дайджест;
- `--filter`, `-f` — фильтрация выходных данных на основе предоставленных условий;
- `--format` — форматирование вывода с использованием пользовательского шаблона: `'table'` : вывод в формате таблицы с заголовками столбцов (по умолчанию) `'table TEMPLATE'` ;
- `--no-trunc` — не обрезать вывод;
- `--quiet`, `-q` — показывать только идентификаторы образов.

Примеры использования команды:

1. Вывод всех образов:

```
docker images
```

```
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
busybox       latest    7cfbbec8963d   7 days ago      4.86MB
ubuntu        latest    08d22c0ceb15   2 weeks ago     7.8MB
hello-world   latest    feb5d9fea6a5   18 months ago   13.3kB
```

2. Вывод образов из репозитория ubuntu:

```
docker images ubuntu
```

```
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       latest    08d22c0ceb15   2 weeks ago   77.8MB
```

3. Вывод образов, соответствующих определенному тегу:

```
docker images hello-world:latest
```

```
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    feb5d9fea6a5   18 months ago   13.3kB
```

2.5.5 Docker create

Команда `docker container create` (или сокращенно: `docker create`) создает новый контейнер из указанного образа, не запуская его.

При создании контейнера демон `docker` создает записываемый слой контейнера поверх указанного образа и подготавливает его к выполнению указанной команды. Идентификатор контейнера затем выводится в `STDOUT`. Действия похожи на

выполнение команды `docker run -d`, за исключением того, что контейнер никогда не запускается. Затем можно использовать команду `docker container start` (или сокращенно: `docker start`) для запуска контейнера в любой момент.

Синтаксис команды:

```
docker create [<опции>] <образ> [<команда>] [<аргументы>...]
```

Основные опции команды:

- `--attach`, `-a` — позволяет манипулировать вводом и выводом по мере необходимости, для вывода информации используется стандартный ввод/вывод контейнера `STDIN`, `STDOUT` и `STDERR`.
- `--cpus` — определяет количество процессоров;
- `--device` — добавить хост-устройство в контейнер;
- `--hostname`, `-h` — имя хоста контейнера;
- `--interactive`, `-i` — оставить `STDIN` открытым без присоединения к терминалу;
- `--memory`, `-m` — лимит памяти. Данный параметр устанавливает верхний предел памяти, доступной для контейнера. Предел задается в контрольной группе, и приложения в контейнере могут запрашивать ее по адресу `/sys/fs/cgroup/memory/memory.limit_in_bytes`;
- `--rm` — автоматическое удаление контейнер после его выхода;
- `--tty`, `-t` — выделить псевдотерминал;
- `--user`, `-u` — имя пользователя или `UID` (формат: `<name|uid>[:<group|gid>]`);
- `--volume`, `-v` — примонтировать том;
- `--volumes-from` — смонтировать тома из указанного(ых) контейнера(ов);
- `--workdir`, `-w` — задать рабочий каталог внутри контейнера.

Примеры использования команды:

1. Создание интерактивного контейнера с прикрепленным псевдотерминалом, запуск контейнера и последующее подключение к нему:

```
docker container create -i -t --name new_container ubuntu
```

```
6d8af538ec541dd581ebc2a24153a28329acb5268abe5ef868c1f1a261221752
```

```
docker container start --attach -i mycontainer
/ # echo hello world
```

```
hello world
```

Вышеуказанные команды эквивалентны реализации через `docker run`:

```
docker run -it --name mycontainer2 ubuntu
/ # echo hello world
```

```
hello world
```

2. Тома контейнеров инициализируются на этапе создания `docker` (т.е. при запуске `docker`). Создадим том `data` в контейнере, а затем попробуем использовать его из другого контейнера:

```
docker create -v /data --name data ubuntu
```

```
240633dfbb98128fa77473d3d9018f6123b99c454b3251427ae190a7d951ad57
```

```
docker run --rm --volumes-from data ubuntu ls -la /data
```

```
total 8
drwxr-xr-x  2 root root 4096 Dec  5 04:10 .
drwxr-xr-x 48 root root 4096 Dec  5 04:11 ..
```

2.5.6 Docker run

Команда `docker run` используется для создания и запуска контейнера из образа. Может принимать несколько аргументов, таких как имя образа, параметры запуска контейнера и команду, которую необходимо выполнить внутри контейнера.

Синтаксис команды:

```
docker run [<опции>] <образ> [<команды>] [<аргументы>...]
```

Основные опции команды:

- `--attach`, `-a` — позволяет манипулировать вводом и выводом по мере необходимости, для вывода информации используется стандартный ввод/вывод контейнера `STDIN`, `STDOUT` и `STDERR`.
- `--cpus` — определяет количество процессоров;
- `--detach`, `-d` — запустить контейнер в фоновом режиме и напечатать `ID` контейнера.
- `--device` — добавить хост-устройство в контейнер.
- `--hostname`, `-h` — имя хоста контейнера;
- `--interactive`, `-i` — оставить `STDIN` открытым без присоединения к терминалу;
- `--memory`, `-m` — лимит памяти. Данный параметр устанавливает верхний предел памяти, доступной для контейнера. Предел задается в контрольной группе, и приложения в контейнере могут запрашивать ее по адресу `/sys/fs/cgroup/memory/memory.limit_in_bytes`;
- `--privileged` — предоставление расширенных прав контейнеру;
- `--rm` — автоматическое удаление контейнер после его выхода;
- `--tty`, `-t` — выделить псевдотерминал;
- `--user`, `-u` — имя пользователя или `UID` (формат: `<name|uid>[:<group|gid>]`);
- `--volume`, `-v` — примонтировать том;
- `--workdir`, `-w` — задать рабочий каталог внутри контейнера.

Примеры использования параметров:

1. Передача данных на стандартный ввод контейнера:

```
echo "test"| docker run -i -a stdin ubuntu cat -
```

2. Запуск контейнера в фоновом режиме и вывод ID контейнера. Например:

```
docker run -d ubuntu
```

```
b85f949c5d1e593fcf732db410502e999339549d1175947a0467652e7053c133
```

3. Добавление в непривилегированный контейнер (без использования флага --privileged) блочного устройства хранения данных, loop-устройства и аудиоустройства и предоставление приложению прямого доступа к ним.

```
docker run -it --rm
--device=/dev/sdc:/dev/xvdc
--device=/dev/sdd
--device=/dev/zero:/dev/foobar
ubuntu ls -l /dev/xvdc,sdd,foobar
```

```
brw-rw---- 1 root disk 8, 2 Dec 9 16:05 /dev/xvdc
brw-rw---- 1 root disk 8, 3 Dec 9 16:05 /dev/sdd
crw-rw-rw- 1 root root 1, 5 Dec 9 16:05 /dev/foobar
```

4. Выполнение команды внутри указанного каталога (если путь не существует, он будет создан внутри контейнера):

```
docker run -w /path/to/dir/ -i -t ubuntu pwd
```

2.5.7 Docker start

Команда docker start предназначена для запуска контейнера.

Синтаксис команды:

```
docker start [<опции>] <контейнер> [<контейнер_2>...]
```

Используемые опции команды:

- --attach, -a — прикрепить STDOUT/STDERR и переадресовать сигналы;
- --detach-keys — переопределение последовательности клавиш для отсоединения контейнера;
- --interactive, -i — прикрепить STDIN контейнера.

Пример использования команды:

```
docker start new_container
```

2.5.8 Docker stop

Команда docker stop предназначена для остановки работы контейнера.

Основной процесс внутри контейнера получит SIGTERM, а по истечении времени ожидания — SIGKILL.

Синтаксис команды:

```
docker stop [<опции>] <контейнер> [<контейнер_2>...]
```

Используемые опции команды:

- `--time`, `-t` — время ожидания (в секундах) перед принудительной остановкой контейнера.

Пример использования команды:

```
docker stop new_container
```

2.5.9 Docker pause

Команда `docker pause` приостанавливает все процессы в указанных контейнерах.
Синтаксис команды:

```
docker pause <контейнер> [<контейнер_2>...]
```

Пример использования команды:

```
docker pause new_container
```

2.5.10 Docker login

Для доступа в некоторые централизованные хранилища образов необходимо иметь учетную запись. После регистрации войти в систему можно, используя команду `docker login`.
Синтаксис команды:

```
docker login [<опции>] [<сервер>]
```

Для входа в систему необходимо указать имя пользователя и пароль своей учетной записи. Для доступа к собственным хранилищам образов необходимо указать имя сервера, если сервер не указан, он определяется демоном `docker` по умолчанию.
Основные параметры команды:

- `--password`, `-p <пароль>` — пароль;
- `--password-stdin` — получить пароль со стандартного ввода;
- `--username`, `-u <имя>` — имя пользователя.

Примеры использования команды:
1. Вход в собственное хранилище образов:

```
docker login localhost:8080
```

2. Вход в хранилище образов с использованием стандартного ввода (`stdin`):

```
cat ./pass.txt | docker login -username user -password-stdin
```

где:

- `./pass.txt` — путь к файлу, в котором хранится пароль учетной записи;
- `user` — имя пользователя для входа в хранилище образов.

2.5.11 Docker attach

Команда `docker attach` позволяет прикрепить стандартный ввод (`STDIN`), вывод (`STDOUT`) и вывод ошибок (`STDERR`) локального терминала к контейнеру, используя идентификатор или имя контейнера. Это позволит просматривать его текущий вывод или управлять им в интерактивном режиме, как если бы команды выполнялись непосредственно в локальном терминале.

Для остановки контейнера используется сочетание клавиш `CTRL+C`, которое передаст контейнеру сигнал `SIGKILL`. Для остановки контейнера может быть использована команда `exit`, которая корректно завершает его работу — это наиболее предпочтительный вариант остановки контейнера.

Если контейнер был запущен с параметрами `-i` и `-t`, можно отсоединиться от контейнера и оставить его запущенным, используя последовательность клавиш `CTRL+P` и `CTRL+Q`.

Синтаксис команды:

```
docker attach [<опции>] <контейнер>
```

Основные параметры команды:

- `--detach-keys` — переопределить последовательность клавиш для отсоединения контейнера;
- `--no-stdin` — не присоединять `STDIN`;
- `--sig-proxy` — передать все полученные сигналы процессу. Значение по умолчанию — `true`.

2.5.12 Docker exec

Команда `docker exec` запускает новую команду в работающем контейнере.

Команда, запущенная с помощью `docker exec`, выполняется только во время работы основного процесса контейнера (`PID 1`), и она не перезапускается, если перезапускается контейнер.

Команда запускается в директории контейнера по умолчанию. Если базовый образ имеет пользовательский каталог, указанный директивой `WORKDIR` в Dockerfile, тогда используется данный каталог.

Команда должна быть исполняемым файлом. Команда, заключенная в кавычки, не сработает.

Например, `docker exec -it my_container sh -c "echo a && echo b"` будет выполнена, а `docker exec -it my_container "echo a && echo b"` — нет.

Синтаксис команды:

```
docker exec [<опции>] <контейнер> <команда> [<аргументы>...]
```

Опции команды:

- `-d`, `--detach` — режим отсоединения: выполнение команды в фоновом режиме;
- `--detach-keys <string>` — переопределение последовательности клавиш для отсоединения контейнера;
- `-e`, `--env <list>` — установка переменных окружения;
- `--env-file <list>` — чтение файла с переменными окружения;
- `-i`, `--interactive` — оставить `STDIN` открытым, даже если он не подключен;
- `--privileged` — предоставить расширенные привилегии команде;
- `-t`, `--tty` — выделить псевдотерминал;

- -u, --user <string> — имя пользователя или UID (формат: <name|uid> [:<group|gid>]);
- -w, --workdir <string> — рабочая директория внутри контейнера.

Примеры использования команды:

1. Создание нового файла /tmp/execWorks внутри работающего контейнера mycontainer в фоновом режиме:

```
docker exec -d mycontainer touch /tmp/execWorks
```

2. Запуск нового сеанса оболочки в контейнере mycontainer:

```
docker exec -it mycontainer sh
```

2.5.13 Docker logs

Команда docker logs предназначена для вывода журналов контейнера.

Синтаксис команды:

```
docker logs [<опции>] <контейнер>
```

Используемые параметры команды:

- --details — показать дополнительные детали, предоставляемые в журнале;
- -f, --follow — следить за выводом журналов;
- --since <string> — показывать записи с указанной временной метки (например, 2022-03-01T13:23:37Z) или с определенного промежутка времени (например, 42m — отобразит записи журнала за последние 42 минуты);
- -n, --tail <string> — количество строк для отображения записей с конца журнала (по умолчанию "все");
- -t, --timestamps — показать временные метки;
- --until <string> — показать записи до указанной временной метки (например, 2022-03-01T13:23:37Z) или до определенного промежутка времени (например, 42m — отобразит записи, сделанные 42 минуты назад и ранее).

Примеры использования команды:

```
docker run --name test -d busybox sh -c "while true; do $(echo
date); sleep 1; done"
```

```
Tue Dec 24 08:16:28 UTC 2024
```

```
docker logs -f --until=2s test
```

```
Tue Dec 24 08:16:28 UTC 2024
Tue Dec 24 08:16:29 UTC 2024
Tue Dec 24 08:16:30 UTC 2024
Tue Dec 24 08:16:31 UTC 2024
```

2.5.14 Docker stats

Команда `docker stats` предназначена для отображения статистики использования ресурсов контейнеров в реальном времени. Для того чтобы ограничить данные одним или несколькими конкретными контейнерами, необходимо указать список имен или идентификаторов контейнеров, разделенных пробелом. Также можно указать остановленный контейнер, но остановленные контейнеры не возвращают никаких данных.

Синтаксис команды:

```
docker stats [<опции>] [<контейнер>...]
```

Используемые опции команды:

- `--all`, `-a` — показать все контейнеры (по умолчанию отображаются только запущенные);
- `--format` — форматирование вывода с использованием пользовательского шаблона: `'table'` : вывод в формате таблицы с заголовками столбцов (по умолчанию) `'table TEMPLATE'`;
- `--no-stream` — отключить потоковую статистику и получать только первый результат;
- `--no-trunc` — не обрезать вывод.

Примеры использования команды:

1. Вывод общей статистики использования ресурсов.

```
docker stats
```

```
CONTAINER ID   NAME         CPU %     MEM USAGE / LIMIT   MEM %
NET I/O       BLOCK I/O   PIDS
a94cd881360f   test_con3   0.17%     416KiB / 15.63GiB   0.00%
2.1kB / 0B    0B / 0B     1
4bc7571b9508   test_con2   0.15%     424KiB / 15.63GiB   0.00%
2.1kB / 0B    0B / 0B     1
9f426fb78c27   test_con1   0.15%     420KiB / 15.63GiB   0.00%
2.71kB / 0B   0B / 0B     1
```

2. Вывод статистики контейнера с именем `test_con2` и получение вывода в `json`-формате.

```
docker stats test_con2 --no-stream --format "{{ json . }}"
```

```
"BlockIO":"0B / 0B", "CPUPerc":"0.20%","Container":"test_con2",
"ID":"4bc7571b9508", "MemPerc":"0.00%","MemUsage":"424KiB /
15.63GiB", "Name":"test_con2", "NetIO":"2.31kB / 0B", "PIDs":"1"
```

3. Вывод статистики для указанных имени и `ID` контейнеров.

```
docker stats 9f426fb78c27 test_con3
```

```
   CONTAINER ID   NAME        CPU %    MEM USAGE / LIMIT   MEM %
NET I/O       BLOCK I/O   PIDS
   9f426fb78c27   test_con1   0.00%    420KiB / 15.63GiB   0.00%
2.71kB / 0B   0B / 0B     1
   a94cd881360f   test_con3   0.16%    416KiB / 15.63GiB   0.00%
2.31kB / 0B   0B / 0B     1
```

2.5.15 Docker tag

Команда `docker tag` предназначена для присвоения определенного имени и, по желанию, тега образу в локальном хранилище.

Синтаксис команды:

```
docker tag SOURCE_IMAGE[:<тег>] TARGET_IMAGE[:<тег>]
```

где `SOURCE_IMAGE` — это имя образа или его `ID`, которому требуется назначить новое имя и/или тег, а `TARGET_IMAGE` — новое имя образа, которое необходимо ему присвоить. Дополнительный параметр `<тег>` может использоваться для указания конкретной версии образа.

Если задано имя собственного хранилища образов, за ним может дополнительно следовать номер порта в формате `:8080`. Имя хранилища должно соответствовать стандартным правилам `DNS`, но не может содержать символы подчеркивания.

Компоненты имени могут содержать строчные буквы, цифры и разделители. Разделитель определяется как точка, один или два знака подчеркивания или один или несколько дефисов. Компонент имени не может начинаться или заканчиваться разделителем.

Имя тега может содержать строчные и прописные буквы, цифры, символы подчеркивания, точки и дефисы. Имя тега не может начинаться с точки или дефиса и может содержать не более 128 символов.

Примеры использования команды:

1. Присвоение образу с именем myapp тега `v1.0` и последующее переименование в `myregistry/myapp:v1.0`:

```
docker tag myapp myregistry/myapp:v1.0
```

2. Присвоение тега `version1.0` локальному образу с идентификатором `0e557428-3393` и перемещение его в репозиторий `fedora`:

```
docker tag 0e5574283393 fedora/httpd:version1.0
```

2.6 Безопасность в среде контейнеризации

2.6.1 Изоляция контейнеров

Пространства имен (namespaces) в среде контейнеризации обеспечивают изоляцию запущенных процессов, ограничивая их доступ к системным ресурсам. Главная задача каждого пространства имен — отделить определенный глобальный ресурс хостовой системы в некоторую среду, которая заставит считать работающие процессы, что они выполняются в своем собственном экземпляре глобального ресурса.

В среде контейнеризации обеспечиваются следующие механизмы изоляции контейнеров:

- изоляция пространств идентификаторов процессов контейнеров (PID);
- изоляция пространств имен для межпроцессного взаимодействия контейнеров (IPC);
- изоляция пространств имен для пользователей и групп контейнеров (USER);
- изоляция пространств имен хостов и доменов контейнеров (UTS);
- изоляция сетевых пространств имен контейнеров (NETWORK);
- изоляция пространств имен для иерархии каталогов контейнеров (MOUNT).

Пространства имен идентификаторов процессов контейнеров (**PID namespaces**) изолируют пространство идентификаторов процессов, следовательно, процессы в разных пространствах могут иметь одинаковые `PID`. Одно из основных преимуществ **PID namespaces** заключается в том, что контейнеры можно перемещать между хостами, сохраняя при этом одинаковые идентификаторы процессов внутри контейнера.

PID namespaces также позволяют каждому контейнеру иметь свой собственный инициализирующий (родительский) процесс с `PID 1`, который управляет различными задачами инициализации системы. При завершении инициализирующего процесса ядро завершает все процессы, принадлежащие одному контейнеру, через сигнал `SIGKILL`.

Пространства имен для межпроцессного взаимодействия контейнеров (**IPC namespaces**) изолируют объекты `IPC System V` и очереди сообщений `POSIX`.

`System V IPC` включает в себя семафоры, разделяемую память и очереди сообщений. Каждый ресурс межпроцессного взаимодействия должен иметь свой уникальный идентификатор (`ID`). Это позволяет процессам, взаимодействующим между собой, обращаться к общему ресурсу. Объекты, созданные в этом пространстве имён, видны всем процессам, состоящим в этом пространстве, и не видны всем остальным процессам в других пространствах имен.

Пространства имён пользователей (**USER namespaces**) изолируют `ID` пользователей и групп, корневой каталог, ключи и права. Идентификаторы пользователя и группы процесса могут отличаться внутри и вне пространства имён пользователей. В частности, процесс может иметь непривилегированный ID пользователя вне пространства и, в то же время, иметь `ID` суперпользователя `root` внутри пространства.

Сетевые пространства имен (**NETWORK namespaces**) обеспечивают изоляцию системных сетевых ресурсов. Таким образом, каждое сетевое пространство имен имеет свои собственные сетевые устройства, `IP`-адреса, таблицы `IP`-маршрутизации, межсетевые экраны, номера портов и т. д.

Физические сетевые устройства могут принадлежать только одному пространству. При этом каждое пространство имён может иметь одно или несколько виртуальных устройств, для обеспечения доступа во внешнюю сеть между физическим и виртуальным устройством из разных пространств создается мост — туннель между разными пространствами имён сети.

Когда последний процесс в пространстве завершается, физическое сетевое устройство возвращается не в родительское пространство, а в пространство хостовой машины.

Пространства имен хостов и доменов (**UTS namespaces**) изолируют два системных идентификатора — имя узла и имя домена. В контексте контейнеров функция пространств имен UTS позволяет каждому контейнеру иметь собственное имя хоста

и доменное имя NIS.

Пространства имен для иерархии каталогов (**MOUNT namespaces**) изолируют набор точек монтирования файловой системы, видимых группой процессов. Таким образом, процессы в разных пространствах имен монтирования могут иметь разные представления иерархии файловой системы.

2.6.2 Выявление уязвимостей в образах контейнеров

Для выявления уязвимостей в образах контейнеров совместно со встроенными средствами защиты ОС применяется сертифицированное программное обеспечение для анализа защищённости и управления ИБ – RedCheck.

Сканер безопасности RedCheck позволяет проводить комплексный аудит безопасности для образов и контейнеров, реализованных на базе платформы контейнеризации Docker и Kubernetes. В рамках данной функции доступны проверки на уязвимости, критичные неустановленные обновления безопасности, неверные настройки параметров конфигураций и пр.

Сканирование уязвимостей выполняется с использованием файла базы данных уязвимостей, рекомендуемого ФСТЭК России и содержащего сведения об уязвимостях банка данных угроз безопасности информации ФСТЭК России.

Сканирование образов контейнеров необходимо выполнять каждый раз перед установкой и запуском образов контейнеров в информационной (автоматизированной) системе. При обнаружении уязвимостей критического и высокого уровня опасности дальнейшее использование образа контейнера запрещено.

Для проверки docker-образа на уязвимости и последующего безопасного создания контейнера необходимо использовать команду вида:

```
IMAGE_NAME="<имя_образа_контейнера>" && \
REPORT=$(trivy-check-image "$IMAGE_NAME" 2>/dev/null) && \
FIRST_PART=$(echo "$REPORT" | sed 's/;.*//') && \
if echo "$FIRST_PART" | grep -q "уязвимостей не обнаружено"; \
then docker pull "$IMAGE_NAME"; \
else if echo "$FIRST_PART" | grep -q \
"критичных - 0, высокого уровня - 0"; \
then docker pull "$IMAGE_NAME"; \
else echo "Уязвимости обнаружены!"; false; fi; fi
```

Для безопасного создания контейнера Kubernetes необходимо использовать команду вида:

```
YAML_FILE="<файл_манифеста>" &&
IMAGE_NAME=$(grep -E '^\s*image:' "$YAML_FILE" | head -n1 | awk \
'print $2') && \
REPORT=$(trivy-check-image "$IMAGE_NAME" 2>/dev/null) && \
FIRST_PART=$(echo "$REPORT" | sed 's/;.*//') && \
if echo "$FIRST_PART" | grep -q "уязвимостей не обнаружено"; \
then kubectl apply -f "$YAML_FILE"; \
else if echo "$FIRST_PART" | grep -q \
"критичных - 0, высокого уровня - 0"; \
then kubectl apply -f "$YAML_FILE"; \
```

```
  else echo "Уязвимости обнаружены!"; false; fi; fi
```

Сканирование образов контейнеров на наличие известных уязвимостей образов контейнеров необходимо выполнять не реже 1 раза в неделю. Расписание сканирования необходимо настроить в RedCheck.

Результат сканирования уязвимостей включает в себя две основные вкладки с требуемой информацией:

- «Информация по образам» – содержит информацию о найденных образах на сканируемом хосте:
  - Название репозитория;
  - Количество уязвимостей;
  - Количество контейнеров;
  - Тег образа;
  - Размер образа;
  - ОС и архитектура;
  - Дата создания образа;
  - Имя образа (хеш).
- «Результат» – содержит информацию о найденных в образах уязвимостях:
  - ALTX ID – внутренний идентификатор уязвимости;
  - Риск – уровень критичности;
  - OVAL – ссылка на страницу уязвимости в `OVALdb`;
  - Ссылки – страницы уязвимости в различных базах данных уязвимостей;
  - Продукты – ПО, подверженное уязвимости;
  - Детализация – файлы, подверженные уязвимости.

Результаты сканирования сохраняются в файл формата `XML` – `results.xml` в каталог `/var/opt/scan-service/jobs/executionId/uuid/`.

Оповещение о выявленных уязвимостях в образах контейнеров разработчика образов контейнеров и администратора безопасности информационной (автоматизированной) системы производится путем отправки отчета о сканировании на указанную при настройке задания сканера RedCheck электронную почту.

2.6.3 Проверка корректности конфигурации контейнеров

По умолчанию в контейнере недоступны устройства хостовой системы. Для подключения необходимых устройств на этапе запуска (создания или обновления конфигурации) контейнера требуется указать значение аргумента `--device`.

Для получения списка устройств контейнера необходимо запустить контейнер с подключенным устройством:

- для подключения жесткого диска:

```
   docker run -it --rm --name test --device=/dev/sda
registry.red-soft.ru/ubi8/ubi
```

- для подключения периферийного устройства, например `CD-ROM`:

```
   docker run -it --rm --name test --device=/dev/cdrom
registry.red-soft.ru/ubi8/ubi
```

Далее необходимо проверить список доступных контейнеру устройств:

```
ls /dev
```

```
console  fd    mqueue  ptmx  random  shm     stdin   tty     zero
core     full  null    pts   sda     stderr  stdout  urandom
```

Для отображения информации о подключенных дисках в контейнере используются команды `fdisk` (для жестких дисков) и `blkid` (для блочных устройств). Например:

```
fdisk -l
```

```
Disk /dev/sda: 15.15 GiB, 16265093120 bytes, 31767760 sectors
Disk model: VBOX HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x2e69b634

Device     Boot   Start       End   Sectors   Size Id Type
/dev/sda1  *       2048   2099199   2097152     1G 83 Linux
/dev/sda2       2099200  31766527  29667328  14.1G 8e Linux LVM
```

```
blkid
```

```
/dev/cdrom: BLOCK_SIZE="2048" UUID="2022-10-27-17-32-59-00"
LABEL="redos-8-x86_64" TYPE="iso9660" PTUUID="0b10fcfb" PTTYPE="dos"
```

Для получения списка доступных устройств на хостовой ОС выполните:

```
ls /dev
```

Для ограничения в контейнере количества операций ввода/вывода в секунду необходимо при запуске (создании или обновлении конфигурации) контейнера указать параметр `--device-write-iops` (или `--device-read-iops`). Например, установим ограничение 10 операций ввода/вывода в секунду:

```
docker run -it --rm --name test --device-write-iops /dev/sda:10
registry.red-soft.ru/ubi8/ubi
```

Для отслеживания статистики в контейнере выполните:

```
time dd if=/dev/zero of=test.out bs=1M count=1024 oflag=direct
```

```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 204.826 s, 5.2 MB/s
```

```
real    3m24.836s
user    0m0.025s
sys     0m0.400s
```

При запуске контейнера без ограничения количества операций ввода/вывода статистика выглядит следующим образом:

```
time dd if=/dev/zero of=test.out bs=1M count=1024 oflag=direct
```

```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 3.61733 s, 297 MB/s

real    0m3.623s
user    0m0.000s
sys     0m0.293s
```

Для монтирования файловой системы хостовой ОС необходимо использовать параметр назначения прав только для чтения — `ro`. Например, создайте контейнер и подключите к нему корневую директорию хостовой ОС командой:

```
docker run -it --rm --name test -v /:/mnt/:ro registry.red-soft.
ru/ubi8/ubi
```

Для вывода содержимого каталога контейнера `/mnt` выполните:

```
ls -l /mnt
```

```
total 68
lrwxrwxrwx.   1 root root     7  Dec  31   2021  bin -> usr/bin
dr-xr-xr-x.   6 root root  4096  Dec  13  08:10  boot
drwxr-xr-x.  20 root root 14180  Dec  16  12:25  dev
drwxr-xr-x.  10 root root 12288  Dec  16  12:25  etc
drwxr-xr-x.   6 root root  4096  Dec  16  07:13  home
```

При попытке создать какой-либо фал в данном каталоге будет выведено сообщение об ошибке вида:

```
touch /mnt/test_file
```

```
touch: cannot touch '/mnt/test_file': Read-only file system
```

2.6.4 Контроль целостности контейнеров и их образов

Контроль целостности образов контейнеров и их исполняемых файлов производится инструментом хостовой ОС — `IMA`, контроль целостности сведений о событиях безопасности осуществляется средством верификации целостности хостовой ОС — `afick`. Подробную информацию об инструментах см. в подразделе «Afick — верификация целостности» и подразделе «Контроль целостности запускаемых компонентов

программного обеспечения» Руководства администратора. Часть 1.

По умолчанию из-за высокой нагрузки на ЭВМ функция контроля целостности выключена. Включение производится в файле настроек `daemon.json` установкой значения параметра `disable-validation` в `false`. Проверка целостности по расписанию (1 раз в неделю) по умолчанию также отключена по тем же причинам. Для включения проверки целостности образов контейнеров и файла конфигурации необходимо выполнить:

```
systemctl enable docker-validate.timer --now
```

Для изменения настроек расписания проверки целостности образов контейнеров и файла конфигурации средства контейнеризации необходимо изменить настройки таймера в файле `/usr/lib/systemd/system/docker-validate.timer` в секции `[Timer]`.

По умолчанию файл имеет следующее содержимое:

```
[Unit]
Description=Checking the integrity of container images and confi-
guration file once a week
ConditionVirtualization=!container
ConditionPathExists=/var/run/docker.sock
ConditionFileIsExecutable=/usr/bin/curl

[Timer]
OnCalendar=weekly
Persistent=true

[Install]
WantedBy=timers.target
```

Примечание. Не рекомендуется осуществлять проверку контроля целостности образов контейнеров и файла конфигурации средства контейнеризации чаще 1 раза в час. Индивидуальные настройки зависят от количества установленных образов и вычислительной мощности отдельно взятой ЭВМ.

После внесения каких-либо изменений в файл настройки таймера необходимо его перезапустить, чтобы внесенные изменения вступили в силу:

```
systemctl restart docker-validate.timer
```

Отключение проверки целостности образов по расписанию реализуется выполнением команды:

```
systemctl disable docker-validate.timer --now
```

При попытке создать (запустить) контейнер из образа, целостность которого нарушена, будет выведено сообщение об ошибке примерно следующего вида, а запуск (создание) будет отменено:

```
docker: Error response from daemon: The image is corupted!
See 'docker run --help'
```

В журнале хостовой ОС будет зафиксирована попытка запуска контейнера из поврежденного образа. Запись в журнале будет иметь примерно следующий вид:

```
journalctl -xe -u docker
```

```
...
май 16 13:08:52 localhost.localdomain dockerd[10789]: time="2025
-05-16T13:08:52.600838411+03:00" level=error msg="The image is corup-
ted!" id=1c8b8da7-4f6c-4b1b-b56c-3b6a5e699135 image=70ea72b3cd35fd95
de1b279dfa3f5b50a49e191246ce880c935bc74af6535e8c type="Vadlidate
image" user=1001
...
```

Также средством контейнеризации осуществляется контроль целостности параметров настройки с помощью инструмента хостовой ОС — `IMA`.

В журнале безопасности хостовой ОС будут зарегистрированы соответствующие записи о нарушении целостности файла настроек средства контейнеризации:

```
journalctl -xe -u docker
```

```
...
май 16 16:40:23 localhost.localdomain dockerd[857833]: time="2025
-05-16T16:40:23.638012556+03:00" level=info msg="The settings file
has passed integrity control" id=642490c2-f3ab-4dcd-a933-3490ffba86-
dc result=OK type="Vadlidate config file" user=
...
```

2.6.5 Регистрация событий безопасности

Регистрация событий безопасности производится средствами хостовой ОС.

В журнале безопасности регистрируются такие виды событий, как:

- создание, модификация и удаление образов контейнеров;
- запуск и остановка контейнеров с указанием причины остановки;
- модификация запускаемых контейнеров;
- факты нарушения целостности объектов контроля.

Просмотр зарегистрированных действий осуществляется с помощью команды (с правами суперпользователя):

```
journalctl -xe -u docker
```

События по созданию, модификации и удалению образов отображаются в виде следующих записей:

```
май 19 10:33:18 localhost.localdomain dockerd[7669]: time="2025-
05-19T10:33:18.023189296+03:00" level=info msg="The image was crea-
ted." id=6da4d000-7261-441b-99eb-59098d519138 image=ad4f35aad12f1e3-
caefdd36b29e03b62e049053ae7b91f7994143dc57259c48e result=OK type=
"Build image" user=1001
...
```

```
   май 19 10:35:02 localhost.localdomain dockerd[7669]: time="2025-
05-19T10:35:02.917209586+03:00" level=info msg="The image was commi-
ted." id=153d2f86-461d-49a2-bc9a-19c8ed94059c image=3d9cd2346b53888-
f0bcb77776b72f331f82e9fd0035963439e7249a834fd8502 parent=ad4f35aad1-
2f1e3caefdd36b29e03b62e049053ae7b91f7994143dc57259c48e result=OK
type="Commit image"user=1001
   ...
   май 19 10:37:09 localhost.localdomain dockerd[7669]: time="2025-
05-19T10:37:09.488058315+03:00" level=warning msg="The image was de-
leted." id=25ffc1e4-d765-44a7-bd27-1ea1b758dc45 image=ad4f35aad12f1-
e3caefdd36b29e03b62e049053ae7b91f7994143dc57259c48e result=OK type=
"Delete image" user=1001
   май 19 10:37:09 localhost.localdomain dockerd[7669]: time="2025-
05-19T10:37:09.682221342+03:00" level=warning msg="The image was de-
leted." id=3ed4b675-d8ad-4ac6-b246-991ec1eb3ad6 image=3d9cd2346b53-
888f0bcb77776b72f331f82e9fd0035963439e7249a834fd8502 result=OK type=
"Delete image" user=1001
   ...
```

Записи о запуске и остановке контейнеров выглядят следующим образом:

```
   ...
   май 19 10:50:08 localhost.localdomain dockerd[7669]: time="2025-
05-19T10:50:08.980129228+03:00" level=info msg="The container has
been created." container=b630ae774b7495b52b2ad517dcbc0263ccc1f8bb68-
173fc77d5b85e7128e62cb id=c4f79a5e-8ac2-45ab-a459-22e65513731f
result=OK type="Create container" user=1001
   май 19 10:50:12 localhost.localdomain dockerd[7669]: time="2025-
05-19T10:50:12.046570726+03:00"level=info msg="The container has
been started." container=b630ae774b7495b52b2ad517dcbc0263ccc1f8bb68-
173fc77d5b85e7128e62cb id=87536452-badf-43a0-8e84-afe9f10bbe63
result=OK type="Start container" user=1001
   ...
   май 19 10:52:07 localhost.localdomain dockerd[7669]: time="2025-
05-19T10:52:07.419240037+03:00"level=info msg="The container has
been stopped." container=b630ae774b7495b52b2ad517dcbc0263ccc1f8bb68-
173fc77d5b85e7128e62cb id=8c13117b-7f36-4be9-815e-fed344adc92e rea-
son="Testing reason" result=OK type="Stop container" user=1001
   ...
```

Записи о модификации запущенных контейнеров имеют следующий вид:

```
   ...
   май 19 12:55:57 localhost.localdomain dockerd[7669]: time="2025-
05-19T12:55:57.938404067+03:00" level=info msg="The container has
been created." container=9cd7084fa3261c068353f26d265b4870912e45bb1-
ed5891fcf6c506543b96954 id=8b5369f6-be6d-4c51-b358-fbe528b15258
result=OK type="Create container" user=1001
   май 19 12:56:01 localhost.localdomain dockerd[7669]: time="2025-
05-19T12:56:01.119789828+03:00" level=info msg="The container has
```

```
been started." container=9cd7084fa3261c068353f26d265b4870912e45bb1-
ed5891fcf6c506543b96954 id=98057125-22bc-4def-86ac-e860cdabd867
result=OK type="Start container" user=1001
   май 19 12:57:17 localhost.localdomain dockerd[7669]: time="2025-
05-19T12:57:17.112772813+03:00" level=warning msg="The container has
been updated!" container=9cd7084fa3261c068353f26d265b4870912e45bb1-
ed5891fcf6c506543b96954 id=a34ea823-ac59-47b1-b031-aad18f95492d
result=OK type="Update container" user=1001
   ...
```

Журнал событий безопасности средства контейнеризации доступен только для чтения. Проверку установленных прав на журнал можно произвести командой:

```
ls -l /var/log/docker/docker.log
```

```
-rw-r--r--. 1 root root 3804 май 19 16:49 /var/log/docker/docker.log
```

Стандартная настройка ротации и архивации журнала производится при достижении размера `10 МБ`. Настройка параметров осуществляется только суперпользователем `root` в файле `/etc/logrotate.d/docker`.

Содержимое файла имеет следующий вид:

```
/var/log/docker/docker.log {
  missingok
  maxsize 10M
  compress
  rotate 10
  copytruncate
  delaycompress
  create 0600 root root
}
```

При достижении файлом журнала максимально установленного размера будет выполнено его архивирование. Копия журнала будет сохранена в каталоге `/var/log/docker/` с указанием даты ее создания.

Пример файла:

```
ls -l /var/log/docker
```

```
-rw-r--r--. 1 root root     0 май 20 17:38 docker.log
-rw-r--r--. 1 root root 13497 май 19 14:15 docker.log-20250519
```

3. Платформа Kubernetes

3.1 Основные понятия

Kubernetes — это программная платформа для автоматического управления контейнеризованными приложениями. Kubernetes предлагает базовые механизмы для их развертывания, масштабирования и поддержки.

`Nodes` (узлы, ноды) — физические или виртуальные машины, на которых разворачиваются и запускаются контейнеры с приложениями. Контейнеры на нодах не могут быть запущены напрямую, они должны быть упакованы в структуру более высокого уровня, называемую `pod` (под). В под может быть упакован как один, так и несколько контейнеров. Любые контейнеры в одном поде используют одни и те же ресурсы и локальную сеть.

`Nodes` бывают двух типов:

- `Master` (мастер-нода) — узел, управляющий всем кластером. Он отслеживает остальные ноды и распределяет между ними нагрузку с помощью менеджера контроллеров и планировщика. Обычно `master` занимается только управлением и не берет на себя рабочие нагрузки. Для повышения отказоустойчивости необходимо наличие нескольких мастер-нод.
- `Worker` (рабочие ноды) — узлы, на которых работают контейнеры. В зависимости от параметров ноды (объема памяти и центрального процессора) на одном узле может работать большое количество контейнеров. Чем больше рабочих узлов, тем больше приложений можно запустить. Также количество влияет на отказоустойчивость кластера, потому что при выходе из строя одной ноды нагрузка переносится на другие.

Совокупность нод образует кластер Kubernetes.

3.2 Доступ к среде контейнеризации

Доступ к среде контейнеризации Kubernetes и запуску необходимых для работы служб настраивается на этапе развертывания `master`-ноды (управляющего узла). Подробную информацию о настройке доступа к среде контейнеризации см. в подразделе 3.7 «Настройка Master-node».

3.3 Компоненты Kubernetes

Развертывание среды Kubernetes представляет собой работу с кластером.

Кластер Kubernetes состоит из набора узлов, на которых запускается контейнеризированное программное обеспечение. Кластер должен иметь как минимум один рабочий узел.

На рабочих узлах размещены поды, которые являются компонентами кластера. Главный узел управляет рабочими узлами и подами в кластере. В промышленных средах главный узел обычно запускается на нескольких машинах, а кластер, как правило, развёртывается на нескольких управляющих и рабочих узлах, гарантируя отказоустойчивость и высокую надёжность.

3.3.1 Компоненты управляющего узла

Компоненты управляющего узла отвечают за основные операции кластера (например, планирование), а также обрабатывают события кластера (например, запускают новый под, когда поле `replicas` развертывания не соответствует требуемому количеству реплик).

Компоненты управляющего узла могут быть запущены на любой ноде в кластере. Однако для упрощения работы сценарии настройки обычно запускают все компоненты управляющего узла на одной ноде и в то же время не позволяют запускать пользовательские контейнеры на этой ноде.

kube-apiserver

`kube-apiserver` — это один из ключевых компонентов Kubernetes, который выполняет роль `REST API` сервера для управления кластером. Он обеспечивает взаимодействие между различными компонентами Kubernetes (такими как поды, сервисы, репликации и пр.) и внешними клиентами.

`kube-apiserver` обрабатывает различные события и уведомляет другие компоненты об изменениях в состоянии объектов кластера, обеспечивая тем самым синхронизацию состояния между различными компонентами кластера. `kube-apiserver` также позволяет управлять конфигурациями кластера, включая обновление и удаление объектов.

kube-scheduler

`kube-scheduler` – это компонент панели управления, который отслеживает созданные поды без привязанного узла и выбирает узел, на котором они должны работать.

При планировании развёртывания подов на узлах учитывается множество факторов, включая требования к ресурсам, ограничения, связанные с аппаратными/программными политиками, место расположения данных, предельных сроков.

kube-controller-manager

`kube-controller-manager` – это один из основных компонентов в Kubernetes, который управляет различными контроллерами.

Контроллеры — это циклы управления, которые непрерывно следят за состоянием кластера и, при необходимости, вносят определенные изменения. Каждый контроллер пытается приблизить текущее состояние кластера к желаемому.

В состав `kube-controller-manager` входят следующие типы контроллеров:

- контроллер узла (`node controller`) – отслеживает состояние узлов кластера и реагирует на сбои;
- контроллер репликации (`replication controller`) – отслеживает количество запущенных подов и создает или удаляет их по мере необходимости;
- контроллер конечных точек (`endpoints controller`) – заполняет объект конечных точек (`endpoints`), т.е. связывает сервисы (`services`) и поды (`pods`).
- контроллеры учетных записей и токенов (`account&token controllers`) – создает стандартные учетные записи и токены доступа `API` для новых пространств имен.

3.3.2 Компоненты узла

Описанные далее компоненты работают на каждом узле, поддерживая работу подов и среды выполнения Kubernetes.

kubelet

`kubelet` – агент, работающий на каждом узле в кластере. Он следит за тем, чтобы контейнеры были запущены в поде.

Утилита `kubelet` принимает набор `PodSpecs`, и гарантирует работоспособность и исправность определённых в них контейнеров. Агент `kubelet` не отвечает за контейнеры, не созданные Kubernetes.

kube-proxy

`kube-proxy` — сетевой прокси, работающий на каждом узле в кластере и реализующий часть концепции сетевого сервиса.

`kube-proxy` конфигурирует правила сети на узлах, с помощью которых разрешаются сетевые подключения к подам изнутри и снаружи кластера.

`kube-proxy` использует уровень фильтрации пакетов операционной системы, если он доступен. В противном случае, `kube-proxy` сам обрабатывает передачу сетевого трафика.

Среда выполнения контейнера

Исполняемая среда контейнера — это программа, предназначенная для запуска контейнера в Kubernetes. Kubernetes поддерживает различные среды для запуска контейнеров: `docker`, `containerd`, `CRI-O` и любые реализации Kubernetes `CRI` (Container Runtime Interface).

В рамках настоящего руководства исполняемой средой контейнеров является `containerd`.

3.3.3 Образы контейнеров для создания минимального кластера Kubernetes в закрытом контуре

Таблица 3.1 — Образы контейнеров, необходимые для создания минимального кластера Kubernetes.

Наименование образа контейнера	Назначение
`coredns`	Основной DNS-сервер, отвечающий за разрешение имен внутри кластера Kubernetes.
`etcd`	Строго согласованное распределённое хранилище ключ-значение. В кластере Kubernetes используется для уведомления составных частей и подов/контейнеров об изменении значений ключей.
`flannel-cni-plugin-flannel`	Сетевой плагин, который используется для создания наложенных сетей (`overlay networks`) в Kubernetes.
`flannel-flanneld`	Обеспечивает подключение подов к сети, управляет маршрутизацией и поддерживает сетевые политики, обеспечивая надежную работу приложений.
`kube-apiserver`	API-сервер кластера Kubernetes.
`kube-controller-manager`	Набор различных контроллеров кластера Kubernetes – необходим для поддержания частей кластера в требуемом состоянии.
`kube-scheduller`	Планировщик, который решает, какой узел соответствует требованиям для запуска пода.
`pause`	Контейнер-шаблон, резервирующий `namespaces`, которые являются общими для всех контейнеров внутри пода.
`registry`	Основное хранилище Docker-образов (не обязателен для запуска кластера).

3.4 Управление кластерами

3.4.1 Управление кластерами с помощью kubeadm

`Kubeadm` – это инструмент командной строки, разработанный для установки и настройки кластера Kubernetes. Он автоматизирует процесс создания и управления кластером, включая инициализацию управляющих узлов и присоединение рабочих узлов к кластеру.

Синтаксис утилиты имеет вид:

```
kubeadm <команда>
```

где `<команда>` – определяет выполняемую операцию с ресурсами кластера, например, `init`, `join`, `reset`.

Основные команды утилиты:

- `init` – инициализация управляющего узла, настройка необходимых для работы параметров, создание файлов конфигураци;

- `join` – присоединение рабочей ноды к кластеру;
- `kubeconfig` – работа с файлами `kubeconfig`;
- `reset` – сброс настроек кластера, удаление всех конфигураций и данных Kubernetes;
- `token` – управление загрузочными токенами;
- `upgrade` – обновление компонентов кластера до более новой версии.

Далее будут подробнее рассмотрены основные команды управления кластером с помощью `kubeadm`.

С подробной информацией обо всех командах и их работе можно ознакомиться, выполнив:

```
kubeadm --help
```

С подробной информацией об использовании конкретной команды можно ознакомиться, выполнив:

```
kubeadm <команда> --help
```

kubeadm init

Команда `kubeadm init` используется для инициализации кластера и настройки управляющего узла Kubernetes.

Синтаксис команды:

```
kubeadm init [<команда> <опции>]
```

Часто используемые опции команды:

- `--pod-network-cidr=<диапазон_IP>` – адрес внутренней подсети для кластера;
- `--control-plane-endpoint <IP-адрес>` – стабильный `IP`-адрес или `DNS`-имя для управляющего слоя;
- `--token <строка>` – токен, используемый для настройки двунаправленных доверительных отношений между узлами и узлами управляющего слоя. Содержит строчные буквы (`a-z`) и цифры (`0-9`).
  Указывается в формате: `<6_сиволов>.<16_символов>`.

Пример команды инициализации кластера с назначением адреса внутренней подсети:

```
kubeadm init --pod-network-cidr=10.244.0.0/16
```

Подробную информацию об использовании команды см. в справке:

```
kubeadm init --help
```

kubeadm join

Команда `kubeadm join` позволяет инициализировать рабочий узел и присоединить его к кластеру. Команда должна выполняться на рабочем узле, который необходимо присоединить к кластеру.

Для присоединения рабочего узла к кластеру необходимо настроить между ними доверительные отношения. Для этого существует два способа:

1. Использование общего токена с `IP`-адресом сервера `API`.

2. Предоставление специального файла конфигурации.

Синтаксис команды:

```
kubeadm join <опции>
```

Часто используемые опции команды:

- `--token <строка>` – токен для создания доверительных отношений между рабочей нодой и кластером;
- `--discovery-token-ca-cert-hash <хеш-сумма>` – открытый ключ корневого центра сертификации должен соответствовать указанной хеш-сумме;
- `--discovery-file <путь_к_файлу>` – путь к файлу с необходимыми настройками для создания доверительных отношений между рабочей нодой и кластером.

Пример команды присоединения рабочей ноды к кластеру:

```
kubeadm join 10.81.186.86:6443 --token sj5u6t.m8wl18o26fhj4gml \
        --discovery-token-ca-cert-hash sha256:53cbbbaf46036dad\
d5d348eea7b7585d7e7e048a554f7b684a5e5322f9ba498
```

Подробную информацию об использовании команды см. в справке:

```
kubeadm join --help
```

kubeadm token

Команда `kubeadm token` позволяет управлять токенами начальной загрузки, которые используются для настройки двунаправленного доверия между присоединяющимся к кластеру узлом и управляющим узлом.

Для управления токенами могут быть использованы следующие команды:

- `create` – создать токен начальной загрузки на сервере;
- `delete` – удалить токен начальной загрузки на сервере;
- `generate` – сгенерировать и вывести токен начальной загрузки на сервере (без создания);
- `list` – вывести список токенов начальной загрузки.

Основные опции команды:

- `--print-join-command` – вывести полную команду присоединения к кластеру с помощью токена.

Подробную информацию об использовании команды см. в справке:

```
kubeadm token --help
```

3.4.2 Манифесты

Файл манифестов Kubernetes — это один из основных инструментов для создания и настройки компонентов в Kubernetes. Компоненты в кластере Kubernetes можно создавать двумя способами — путем использования команд в консоли либо путем создания файла конфигурации. Преимуществом использования файла манифеста является его простота, наглядность и возможность автоматизации задач в дальнейшем.

Файлы манифестов — это `YAML`- или `JSON`-файлы, которые описывают объекты и их конфигурацию в кластере.

Далее будут рассмотрены основные секции и параметры манифестов.

Секция apiVersion

В секции `apiVersion` обычно указывается версия `API`, которую использует объект. Например:

```
apiVersion: v1
apiVersion: apps/v1
apiVersion: batch/v1
```

Секция kind

Секция `kind` определяет тип объекта, который необходимо создать.

Например:

```
kind: Pod
kind: Deployment
kind: Service
```

Секция metadata

Секция `metadata` содержит данные, которые помогут идентифицировать объект, такие как имя, пространство имен, метки и аннотации:

- `name` – имя объекта;
- `namespace` – пространство имен, в котором будет создан объект (по умолчанию – `default`);
- `labels` – набор меток для организации объектов;
- `annotations` – дополнительная информация об объекте.

Например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
  labels:
    app: sample
  annotations:
    metadata.example.com/title: "my conf File"
    metadata.example.com/author: "Shubham Singh"
    metadata.example.com/properties:
      format: "YAML"
```

Секция specification (spec)

Секция `spec` определяет спецификацию объекта, включая его настройки и параметры:

- для `pod` – указывает контейнеры, их образы, порты и другие параметры;
- для `deployment` – указывает количество реплик, селекторы и шаблон пода.

Например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
      app: nginx
spec:        # спецификация для deployment
  selector:
    matchLabels:
      app: nginx
  replicas: 5
  template:
    metadata:
      labels:
        app: nginx
    spec:      # спецификация для pod
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Секция status

Секция `status` не указывается в файлах-манифестах, т.к. заполняется автоматически Kubernetes после создания объекта.

Например:

```
status:
    availableReplicas: 1
    conditions:
    - lastTransitionTime: "2020-01-24T10:54:597"
        lastUpdateTime: "2020-01-24T10:54:597"
        message: Deployment has minimum availability.
        reason: MinimumReplicasAvailable
         status: "True"
         type: Available
    - lastTransitionTime: "2020-01-24T10:54:567"
         lastUpdateTime: "2020-01-24T10:54:597"
         message: ReplicaSet "nginx-deployment-7d64f4b
         reason: NewReplicaSetAvailable
         status: "True"
         type: Progressing
    observedGeneration: 1
    readyReplicas: 1
    replicas: 1
    updatedReplicas: 1
```

3.4.3 Управление кластером с помощью kubectl

`Kubectl` – это утилита командной строки для управления кластером Kubernetes. `Kubectl` используется для взаимодействия с API-сервером Kubernetes (`kube-apiserver`) и выполнения различных операций, таких как создание, обновление, удаление и просмотр ресурсов кластера.

Синтаксис утилиты имеет вид:

```
kubectl [<команда>] [<тип_ресурса>] [<имя_ресурса>] [<опции>]
```

где:

- `<команда>` – определяет выполняемую операцию с одним или несколькими ресурсами, например, `create`, `get`, `describe`, `delete`;
- `<тип_ресурса>` – определяет тип ресурса. Типы ресурсов не чувствительны к регистру, доступно использование множественной или сокращенной формы. Например, команды ниже выведут один и тот же результат:

  ```
  kubectl get pod pod1
  kubectl get pods pod1
  kubectl get po pod1
  ```

- `<имя_ресурса>` – определяет имя ресурса. Имена чувствительны к регистру. Если имя не указано, то отображаются сведения обо всех ресурсах, например, `kubectl get pods`.
  При выполнении операции с несколькими ресурсами можно группировать ресурсы по типу и имени либо по используемым файлам-манифестам:
  - отображение сведений о ресурсах одного типа (`<тип_1> <имя_ресурса _1> <имя_ресурса_2> ... <имя_ресурса_N>`). Например:

    ```
    kubectl get pod example-pod1 example-pod2
    ```

  - отображение сведений о ресурсах нескольких типов по отдельности (`<тип_1>/<имя_ресурса_1> <тип_2>/<имя_ресурса_2>...<тип_N>/<имя_ресурса_N>`). Например:

    ```
    kubectl get pod/pod1 replicationcontroller/rc1
    ```

  - выбор ресурсов по файлам-манифестам (`-f <файл_1> -f <файл_2> ... -f <файл_N>`). Например:

    ```
    kubectl get pod -f ./pod.yaml
    ```

    Рекомендуется использовать формат файла `YAML` вместо `JSON`, т.к. формат `YAML` является более удобным для пользователей, особенно в конфигурационных файлах.
- `<опции>` – определяет дополнительные флаги. Например, можно использовать флаг `-s` или `--server` для указания адреса и порта `API`-сервера Kubernetes.

Примечание. Переданные `kubectl` опции переопределят значения по умолчанию и связанные переменные окружения.

Основные команды управления кластерами:

- `kubectl apply` – внести изменения в конфигурацию ресурса из файла или потока стандартного ввода (`stdin`);
- `kubectl attach` – подключиться к запущенному контейнеру для просмотра потока вывода либо для работы с контейнером (`stdin`);
- `kubectl cluster-info` – показать информацию о главном узле и сервисах в кластере;
- `kubectl config` – управление конфигурацией клиента `kubectl`;
- `kubectl create` – создать один или несколько ресурсов из файла или из потока стандартного ввода;
- `kubectl delete` – удалить ресурсы из файла, потока `stdin`, с помощью меток, имен или ресурсов;
- `kubectl descrube` – показать подробное состояние одного или нескольких ресурсов;
- `kubectl edit` – отредактировать и обновить определение одного или нескольких ресурсов на сервере, используя редактор по умолчанию;
- `kubectl exec` – выполнить команду в контейнере пода;
- `kubectl get` – вывести один или несколько ресурсов;
- `kubectl logs` – вывести логи контейнера в поде;
- `kubectl run` – запустить указанный образ в кластере.

Далее будут подробнее рассмотрены основные команды управления кластером.

С подробной информацией обо всех командах и их работе можно ознакомиться, выполнив:

```
kubectl --help
```

С подробной информацией об использовании конкретной команды можно ознакомиться, выполнив:

```
kubectl <команда> --help
```

kubectl get

Команда `kubectl get` используется для получения информации о ресурсах. Выведенный список ресурсов можно отфильтровать, используя дополнительные опции.

Синтаксис команды:

```
kubectl get <ресурс> [<опции>]
```

Часто используемые опции команды:

- `-f <имя_файла>` – имя файла конфигурации, определяющего необходимый ресурс;
- `--sort-by=<поле>` – сортировка вывода по указанному полю;
- `-o, --output='<формат>'` – вывод информации о ресурсе в указанном формате (`json`, `yaml` и пр.).

Для вывода информации о доступных подах выполните:

```
kubectl get pods
```

Для вывода информации о доступных нодах выполните:

```
kubectl get nodes
```

Подробную информацию об использовании команды см. в справке:

```
kubectl get --help
```

kubectl create

Команда `kubectl create` используется для создания ресурса из файла-манифеста или стандартного ввода.

Синтаксис команды:

```
kubectl create -f <файл> [<опции>]
```

Часто используемые опции команды:

- -f <имя_файла> – имя файла конфигурации, необходимого для создания ресурса.

Подробную информацию об использовании команды см. в справке:

```
kubectl create --help
```

kubectl apply

Команда `kubectl apply` используется для применения конфигурации к ресурсу по его имени или стандартному вводу.

Синтаксис команды:

```
kubectl apply -f <имя_файла> [<опции>]
```

Часто используемые опции команды:

- `-f <имя_файла>` – имя файла конфигурации в формате `JSON` или `YAML`;
- `-k <каталог>` – имя каталога, в котором хранится файл конфигурации в формате `JSON` или `YAML`.

Подробную информацию об использовании команды см. в справке:

```
kubectl apply --help
```

kubectl attach

Команда `kubectl attach` используется для подключения к запущенному контейнеру.

Синтаксис команды:

```
kubectl attach <ресурс> -c <имя_контейнера>
```

Часто используемые опции команды:

- `-c, --container=<имя_контейнера>` – имя комнтейнера. Если параметр отсутствует, будет выбран первый контейнер в поде;
- `--pod-running-timeout=<время>` – время ожидания (например, `5s`, `2m`, `3h`), пока не заработает хотя бы один под;
- `-q, --quiet=false` – показать выходные данные удаленного сеанса.

Подробную информацию об использовании команды см. в справке:

```
kubectl attach --help
```

kubectl describe

Команда `kubectl describe` используется для отображения подробной информации о конкретном ресурсе или группе ресурсов.

Синтаксис:

```
kubectl describe <тип_ресурса> <имя_ресурса>
```

Опции команды:

- `-l, --selector=<имя_параметра>` – фильтровать вывод по указанному значению;
- `--show-events=true` – отображать события, связанные с описываемым объектом.

Подробную информацию об использовании команды см. в справке:

```
kubectl describe --help
```

kubectl exec

Команда `kubectl exec` используется для запуска различных команд в контейнере.

Синтаксис:

```
kubectl exec <тип_ресурса> <имя_ресурса> -- <команда>
```

Например:

```
kubectl exec my-pod -- ls /app
```

Подробную информацию об использовании команды см. в справке:

```
kubectl describe --help
```

kubectl logs

Команда `kubectl logs` используется для отображения журнала контейнера. Если в поде только один контейнер, параметр `-c` допускается не указывать.

Синтаксис:

```
kubectl logs <тип_ресурса> <имя_ресурса> -c <имя_контейнера>
```

Опции команды:

- `-f, --follow=false` – отображение журнала в потоковом режиме;

- `-c <имя_контейнера>` – отображение журнала указанного контейнера;
- `--limit-bytes=0` – максимальный размер отображаемого журнала в байтах. По умолчанию размер не ограничен;
- `--timestamps=false` – отображение временных меток в каждой строке вывода журнала.

Подробную информацию об использовании команды см. в справке:

```
kubectl logs --help
```

kubectl delete

Команда `kubectl delete` используется для удаления ресурсов.

Синтаксис:

```
kubectl delete <тип_ресурса> <имя_ресурса>
```

Опции команды:

- `--cascade='background'` – определяет стратегию каскадного удаления для зависимых объектов. Может принимать значения: `background`, `orphan`, `foreground`.

Подробную информацию об использовании команды см. в справке:

```
kubectl delete --help
```

3.5 Предварительные условия

3.5.1 Требования к среде развертывания

К узлам в среде развертывания `kubernetes` предъявляются следующие минимальные требования:

- 2 ГБ оперативной памяти;
- 2 процессорных ядра;
- 5 ГБ свободного дискового пространства;
- сетевая связанность между узлами (управляющий узел должен иметь возможность соединения с рабочими узлами);
- уникальное имя узла (`hostname`) и `MAC`-адрес для каждого узла;
- открытые порты для планируемых к запуску сервисов, например:
  - порт `80` — для обработки `HTTP`-запросов;
  - порт `6443` — основной порт для взаимодействия с `kube-apiserver`;
  - порт `5000` — для обеспечения сетевого взаимодействия внутри кластера.
- слой контейнеризации (`docker`);
- отключенный файл подкачки (`swap`).

3.5.2 Входные данные

В рамках рассматриваемого далее примера развертывания программной платформы `kubernetes` предполагается, что в работе используется три машины:

- `Registry` – для размещения локального хранилища образов с присвоенным статическим `IP`-адресом `10.0.2.20`,
- `Master` – мастер-нода с присвоенным статическим `IP`-адресом `10.0.2.16`,
- `Worker` – рабочая нода с присвоенным статическим `IP`-адресом `10.0.2.17`,

которые располагаются в закрытой сети `10.0.2.0/24`.

Примечание. Указанные `IP`-адреса необходимо заменить на фактически используемые значения в сети.

3.5.3 Минимально необходимые полномочия контейнеров

Таблица 3.2 — Минимально необходимые полномочия контейнеров.

Наименование образа контейнера	**Минимально необходимые полномочия**		
	Периферийные устройства[1]	Вычислительные ресурсы[2]	Корневая хостовая файловая система[3]
`coredns`, `etcd`, `flannel-cni-plugin-flannel`, `flannel-flanneld`, `kube-apiserver`, `kube-controller-manager`, `kube-proxy`, `kube-scheduller`, `pause`	По умолчанию не используются, права ограничены. Подробнее см. в п. 3.10.3 «Проверка корректности конфигурации контейнеров».	Ограничивать не рекомендуется. Ограничения могут привести к нестабильной работе кластера. Подробнее см. в п. 3.10.3 «Проверка корректности конфигурации контейнеров».	По умолчанию корневая хостовая файловая система не подключена, права ограничены. Подробнее см. в п. 3.10.3 «Проверка корректности конфигурации контейнеров».

Таблица 3.2 — Минимально необходимые полномочия контейнеров (продолжение таблицы).

Наименование образа контейнера	**Минимально необходимые полномочия**		
	Периферийные устройства[1]	Вычислительные ресурсы[2]	Корневая хостовая файловая система[3]
`registry` (если кластер используется в закрытом контуре)	По умолчанию не используются, права ограничены. Подробнее см. в п. 2.6.3 «Проверка корректности конфигурации контейнеров».	Минимальные необходимые вычислительные ресурсы: • ЦП – 1 ядро; • ОЗУ – 100 МБ. Операции ввода-вывода ограничивать не рекомендуется. Для установки ограничений см. п. 2.5.6 «Docker run». Подробнее про общие ограничения см. в п. 2.6.3 «Проверка корректности конфигурации контейнеров».	По умолчанию корневая хостовая файловая система не подключена, права ограничены. Подробнее см. в п. 2.6.3 «Проверка корректности конфигурации контейнеров».

[1]Ограничение прав прикладного программного обеспечения, выполняемого внутри контейнера, на использование периферийных устройств, устройств хранения данных и съемных машинных носителей информации (блочных устройств), входящих в состав информационной (автоматизированной) системы.

[2]Ограничение прав прикладного программного обеспечения, выполняемого внутри контейнера, на использование вычислительных ресурсов (оперативной памяти, операций ввода-вывода за период времени) хостовой операционной системы.

[3]Монтирование корневой файловой системы хостовой операционной системы в режиме "только для чтения".

3.6 Развертывание Docker Registry

3.6.1 Настройка сети

Измените имя хоста:

```
hostnamectl set-hostname registry
```

В файл /etc/hosts внесите данные о registry:

```
nano /etc/hosts
```

```
10.0.2.20 registry
```

3.6.2 Установка docker

Для установки docker выполните команду:

```
dnf install docker-ce
```

Запустите службу docker и добавьте ее в автозагрузку командой:

```
systemctl enable --now docker
```

В файле конфигурации демона docker необходимо указать, что с созданным хранилищем можно взаимодействовать:

```
nano /etc/docker/daemon.json
```

```
{
"insecure-registries": ["10.0.2.20:5000"],
"log-driver": "journald",
"live-restore": true,
"icc": false,
"disable-validation": true
}
```

где:

- 10.0.2.20 – IP-адрес хоста с локальным regisrty (настраиваемый хост);
- 5000 – порт для подключения к локальному regisrty (по умолчанию).

Для сохранения настройки перезапустите службу docker:

```
systemctl daemon-reload
systemctl restart docker
```

3.6.3 Настройка и запуск локального registry

Docker-образ, необходимый для развертывания локального хранилища registry, расположен на установочном диске дистрибутива РЕД ОС. Образ хранится в архиве с расширением .tar.xz в каталоге /images/docker-images/.

Для настройки локального `registry` необходимо:

1. Импортировать образ хранилища `registry`:

```
docker load -i /images/docker-images/registry.tar.xz
```

2. Проверить доступность распакованного образа командой:

```
docker images
```

```
REPOSITORY                                TAG    IMAGE ID
registry.red-soft.ru/redos7c/registry  2.8    ec0246191e3d

CREATED       SIZE
34 hours ago  77MB
```

3. Задать имя для загруженного образа командой:

```
docker tag registry.red-soft.ru/redos7c/registry:2.8 10.0.2.20:\
5000/registry:2.8
```

где:

- `10.0.2.20` — IP-адрес настраиваемого хоста с локальным `registry`;
- `5000` — порт для подключения к локальному `registry` (по умолчанию);
- `registry` — имя образа;
- `2.8` — тег образа.

Примечание. Для удобства администратора все необходимые теги для образов предварительно заданы.

Важно! Во избежание ошибок в процессе развертывания программной платформы `kubernetes` имя и тег распакованного и переименованного образа должны совпадать! ■

4. Запустить контейнер `registry` командой:

```
docker run -d -p 5000:5000 --restart=always --name registry \
10.0.2.20:5000/registry:2.8
```

В случае если образы необходимо хранить отдельно от контейнеров (не в каталоге по умолчанию – `/var/lib/registry`), следует создать новый каталог и примонтировать его при запуске `registry`, заменив каталог по умолчанию:

```
mkdir ~/registry
docker run -d -p 5000:5000 --restart=always --name registry -v \
~/registry:/var/lib/registry 10.0.2.20:5000/registry:2.8
```

Локальное хранилище `docker`-образов настроено. После этого в него можно загружать образы, необходимые для развертывания кластера.

3.6.4 Загрузка образов

Образы, необходимые для развертывания минимального кластера, расположены на установочном диске дистрибутива РЕД ОС. Образы хранятся в архивах с расширением `.tar.xz` в каталоге `/images/docker-images/`.

Для загрузки образов в настроенное локальное хранилище необходимо:

1. Импортировать архивы образов с помощью команды вида:

```
docker load -i /<путь_к_образу>/<имя_образа>.tar.xz
```

Например:

```
docker load -i /images/docker-images/coredns.tar.xz
```

2. Проверить доступность распакованного образа командой:

```
docker images
```

```
REPOSITORY                          TAG           IMAGE ID
registry.red-soft.ru/k8s7c/coredns  v1.8.6        e177830a3a58
registry.red-soft.ru/k8s7c/coredns  1.8.6-240916  e177830a3a58
registry.red-soft.ru/k8s7c/coredns  latest        e177830a3a58

CREATED       SIZE
6 weeks ago   124MB
6 weeks ago   124MB
6 weeks ago   124MB
```

3. Задать новые имена для загруженных образов командой вида:

```
docker tag <имя_распакованного_образа>:<тег> <IP-адрес_registry>\
:<порт>/<имя_образа>:<тег>
```

Например:

```
docker tag registry.red-soft.ru/k8s7c/coredns:v1.8.6 10.0.2.20:\
5000/coredns:v1.8.6
docker tag registry.red-soft.ru/k8s7c/coredns:1.8.6-240916\
10.0.2.20:5000/coredns:1.8.6-240916
docker tag registry.red-soft.ru/k8s7c/coredns:latest 10.0.2.20:\
5000/coredns:latest
```

где:

- `10.0.2.20` — IP-адрес настраиваемого хоста с локальным `registry`;
- `5000` — порт для подключения к локальному `registry` (по умолчанию);
- `coredns` — имя образа;
- `v1.8.6, 1.8.6-240916, latest` — теги образа.

Примечание. Для удобства администратора все необходимые теги для образов предварительно заданы.

Важно! Во избежание ошибок в процессе развертывания программной платформы `kubernetes` имя и тег распакованного и переименованного образа должны совпадать! ■

4. Поместить образы в настроенное локальное хранилище:

```
docker push 10.0.2.20:5000/coredns:v1.8.6
docker push 10.0.2.20:5000/coredns:1.8.6-240916
docker push 10.0.2.20:5000/coredns:latest
```

Примечание. Обратите внимание, что распаковать, задать теги и поместить в локальное хранилище необходимо все образы.

Для проверки работоспособности настроенного локального хранилища попробуйте извлечь из него какой-либо образ, например:

```
docker pull 10.0.2.20:5000/coredns:v1.8.6
```

Если все настроено верно, вывод команды будет иметь примерно следующий вид:

```
v1.8.6: Pulling from coredns
Digest: sha256:1da08db943aa794bc7e2d75b32e8f76ecff0f140cfa6e11\
d1130d7bc15313f35
Status: Image is up to date for 10.0.2.20:5000/coredns:v1.8.6
10.0.2.20:5000/coredns:v1.8.6
```

3.7 Настройка Master-node

1. Для работы `kubelet` отключите `swap`:

- выполните команду для разового отключения:

```
swapoff -a
```

- чтобы `swap` не появился после перезагрузки сервера, выполните команду:

```
swapoff -a && sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

2. Создайте файл для автозагрузки модулей ядра, необходимых для работы сервиса `containerd`:

```
nano /etc/modules-load.d/containerd.conf
```

```
overlay
br_netfilter
```

Загрузите модули в ядро:

```
modprobe overlay
modprobe br_netfilter
```

Проверьте, что данные модули работают:

```
lsmod | egrep "br_netfilter|overlay"
```

3. Создайте конфигурационный файл для работы сети внутри `kubernetes`:

```
nano /etc/sysctl.d/99-kubernetes-cri.conf
```

```
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
```

Примените параметры командой:

```
sysctl --system
```

4. Установите необходимые пакеты:

```
dnf install kubernetes kubernetes-kubeadm cri-tools iproute-tc \
ebtables socat conntrack git curl wget runc containerd
```

Важно! Если настраиваемый хост не имеет доступа в Интернет, потребуется предварительно настроить локальный репозиторий ОС, из которого будет производиться дальнейшая установка необходимых для работы пакетов.

5. Настройте проброс портов в `iptables`:

```
iptables -P FORWARD ACCEPT
```

6. Запустите службу `containerd` и добавьте ее в автозагрузку:

```
systemctl enable --now containerd
```

7. Установите настройки по умолчанию для конфигурации контейнера:

```
containerd config default | sudo tee /etc/containerd/config.toml
```

8. Внесите изменения в файл конфигурации `/etc/containerd/config.toml`:

```
nano /etc/containerd/config.toml
```

8.1 Настройте доступ к локальному `registry`.
Для этого замените строку:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors]
```

на:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."10.0.2.20\
```

```
:5000"]
      endpoint = ["http://10.0.2.20:5000"]
```

8.2 Настройте доступ к контейнеру `pause`. Обратите внимание, что текущая версия Kubernetes (`1.24.17`) совместима с контейнером `pause` только версии `3.7`.

Замените строку:

```
   [plugins."io.containerd.grpc.v1.cri"]
   ...
      sandbox_image = "registry.k8s.io/pause:3.8"
```

на:

```
   [plugins."io.containerd.grpc.v1.cri"]
   ...
      sandbox_image = "10.0.2.20:5000/pause:3.7"
```

8.3 Настройте расположение сетевых плагинов.

Для этого замените строку:

```
   [plugins."io.containerd.grpc.v1.cri".cni]
      bin_dir = "/opt/cni/bin"
```

на

```
   [plugins."io.containerd.grpc.v1.cri".cni]
      bin_dir = "/usr/libexec/cni"
```

8.4 Настройте использование драйвера `cgroup`.

Для это замените строку:

```
   [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.
options]
   ...
   SystemdCgroup = false
```

на:

```
   [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.
options]
   ...
   SystemdCgroup = true
```

Сохраните внесенные изменения.

9. Перезагрузите конфигурации служб:

```
   systemctl daemon-reload
```

10. Перезапустите службу `containerd`:

```
   systemctl restart containerd
```

11. Добавьте службу `kubelet` в автозагрузку:

```
systemctl enable kubelet.service
```

12. Проверьте доступность локального хранилища:

```
ping 10.0.2.20
```

```
PING 10.0.2.20 (10.0.2.20) 56(84) bytes of data.
64 bytes from 10.0.2.20: icmp_seq=1 ttl=64 time=0.248 ms
64 bytes from 10.0.2.20: icmp_seq=2 ttl=64 time=0.312 ms
64 bytes from 10.0.2.20: icmp_seq=3 ttl=64 time=0.391 ms
64 bytes from 10.0.2.20: icmp_seq=4 ttl=64 time=0.413 ms
--- 10.0.2.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3068ms
rtt min/avg/max/mdev = 0.248/0.341/0.413/0.065 ms
```

13. Для инициализации мастер-ноды необходимо:
13.1 Вывести конфигурацию по умолчанию командой:

```
kubeadm config print init-defaults > kubeadm-config.yaml
```

Стандартное содержимое конфигурации будет иметь вид:

```
apiVersion: kubeadm.k8s.io/v1beta3
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 1.2.3.4
  bindPort: 6443
nodeRegistration:
  criSocket: unix:///var/run/containerd/containerd.sock
  imagePullPolicy: IfNotPresent
  name: node
  taints: null
---
apiServer:
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta3
certificatesDir: /etc/kubernetes/pki
clusterName: kubernetes
controllerManager:
```

```
dns:
etcd:
  local:
    dataDir: /var/lib/etcd
imageRepository: registry.k8s.io
kind: ClusterConfiguration
kubernetesVersion: 1.24.0
networking:
  dnsDomain: cluster.local
  serviceSubnet: 10.96.0.0/12
scheduler: {}
```

13.2 Привести содержимое конфигурации к следующему виду:

```
apiVersion: kubeadm.k8s.io/v1beta3
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
  # указать IP-адрес настраиваемой мастер-ноды:
  advertiseAddress: 10.0.2.16
  bindPort: 6443
nodeRegistration:
  criSocket: unix:///var/run/containerd/containerd.sock
  imagePullPolicy: IfNotPresent
  name: node
  taints: null
---
apiServer:
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta3
certificatesDir: /etc/kubernetes/pki
clusterName: kubernetes
controllerManager:
dns:
etcd:
  local:
    dataDir: /var/lib/etcd
    # указать IP-адрес локального registry:
    imageRepository: 10.0.2.20:5000
    # указать <тег> распакованного в п. 3.6.4 образа etcd:
    imageTag: <тег>
```

```
  # указать IP-адрес локального registry:
  imageRepository: 10.0.2.20:5000
  kind: ClusterConfiguration
  # указать версию Kubernetes:
  kubernetesVersion: 1.24.17
  networking:
    # доменное имя для внутреннего DNS в кластере:
    dnsDomain: cluster.local
    # диапазон IP-адресов для сервисов Kubernetes:
    serviceSubnet: 10.96.0.0/12
    # адрес внутренней подсети для кластера:
    podSubnet: 10.244.0.0/16
  scheduler:
```

13.3 Сохранить внесенные в конфигурацию изменения.

13.4 Запустить инициализацию мастер-ноды в одноранговом кластере. Данная команда выполнит начальную настройку и подготовку основного узла кластера.

```
  kubeadm init --config=kubeadm-config.yaml
```

В случае успешной инициализации в конце вывода команды будет отображаться примерно следующее:

```
  Your Kubernetes control-plane has initialized successfully!

  To start using your cluster, you need to run the following as a
regular user:

    mkdir -p $HOME/.kube
    sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
    sudo chown $(id -u):$(id -g) $HOME/.kube/config

  Alternatively, if you are the root user, you can run:

    export KUBECONFIG=/etc/kubernetes/admin.conf

  You should now deploy a pod network to the cluster.
  Run "kubectl apply -f [podnetwork].yaml" with one of the options
listed at:
    https://kubernetes.io/docs/concepts/cluster-administration/addons/

  Then you can join any number of worker nodes by running the
following on each as root:

  kubeadm join 10.0.2.16:6443 --token bh5nw3.uz8a96nhao7u0ue2 \
      --discovery-token-ca-cert-hash sha256:e3623f73dadb313ea38e5f-
04736da6b662925aa78b07d00c50fca6aa0f30e391
```

14. Настройте параметры управления кластером. Настройку можно выполнить как для локального пользователя, так и для суперпользователя `root`.

Для управления кластером от имени локального пользователя выполните команды:

```
mkdir /home/$USER/.kube
cp -i /etc/kubernetes/admin.conf /home/$USER/.kube/config
chown $USER. /home/$USER/.kube /home/$USER/.kube/config
```

Для управления кластером от имени суперпользователя `root` выполните команду:

```
echo "export KUBECONFIG=/etc/kubernetes/admin.conf" >> /root/.bashrc
source /root/.bashrc
export KUBECONFIG=/etc/kubernetes/admin.conf
```

Важно! Если в используемой системе файл, отвечающий за настройки разрешения доменных имен (`/etc/resolv.conf`), располагается по нестандартному пути или имеет нестандартное наименование, скорректируйте путь расположения файла в `configmap`. Для этого выполните команду:

```
kubectl edit configmap coredns -n kube-system
```

и внесите следующие изменения:

```
forward . /<путь_к_файлу>/<имя_файла> {
   max_concurrent 1000
}
```

Примечание. Если настройка производилась для локального пользователя, выйдите из сеанса пользователя `root` командой `exit` и дальнейшие действия по настройке кластера выполняйте от имени локального пользователя.

15. Настройте внутреннюю конфигурацию сети в кластере (в примере используется `flannel`).

15.1 Импортируйте образы на хосте с настроенным локальным `registry`:

```
docker load -i /images/docker-images/flannel-cni-plugin-flannel.tar.xz
docker load -i /images/docker-images/flannel-flanneld.tar.xz
```

15.2 Задайте новые имена для загруженных образов командой вида:

```
docker tag <имя_распакованного_образа>:<тег> <IP-адрес_registry>\
:<порт>/<имя_образа>:<тег>
```

где:

- `<IP-адрес_registry>` — IP-адрес хоста с настроенным локальным `registry`;
- `<порт>` — порт для подключения к локальному `registry` (по умолчанию `5000`);
- `<имя_образа>` — имя образа;
- `<тег>` — тег образа.

Примечание. Для удобства администратора все необходимые теги для образов предварительно заданы.

Важно! Во избежание ошибок в процессе развертывания программной платформы `kubernetes` имя и тег распакованного и переименованного образа должны совпадать! ■

Например:

```
  docker tag registry.red-soft.ru/k8s7c/flannel-cni-plugin-flannel\
:240916 10.0.2.20:5000/flannel-cni-plugin-flannel:240916
  docker tag registry.red-soft.ru/k8s7c/flannel-flanneld:240919 \
10.0.2.20:5000/flannel-flanneld:240919
```

где:

- `10.0.2.20` — IP-адрес хоста с настроенным локальным `registry`;
- `5000` — порт для подключения к локальному `registry` (по умолчанию `5000`);
- `flannel-cni-plugin-flannel, flannel-flanneld` — имена образов;
- `240916, 240919` — теги образов.

15.3 Загрузите распакованные образы в локальное хранилище:

```
  docker push 10.0.2.20:5000/flannel-cni-plugin-flannel:240916
  docker push 10.0.2.20:5000/flannel-flanneld:240919
```

15.4 На `master`-ноде создайте файл конфигурации `kube-flannel.yaml` со следующим содержимым:

```
  apiVersion: v1
  kind: Namespace
  metadata:
    labels:
      k8s-app: flannel
      pod-security.kubernetes.io/enforce: privileged
    name: kube-flannel
  ---
  apiVersion: v1
  kind: ServiceAccount
  metadata:
    labels:
      k8s-app: flannel
    name: flannel
    namespace: kube-flannel
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    labels:
      k8s-app: flannel
```

```
    name: flannel
  rules:
  - apiGroups:
    - ""
    resources:
    - pods
    verbs:
    - get
  - apiGroups:
    - ""
    resources:
    - nodes
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - ""
    resources:
    - nodes/status
    verbs:
    - patch
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    labels:
      k8s-app: flannel
    name: flannel
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: flannel
  subjects:
  - kind: ServiceAccount
    name: flannel
    namespace: kube-flannel
  ---
  apiVersion: v1
  data:
    cni-conf.json: |
      {
        "name": "cbr0",
        "cniVersion": "0.3.1",
        "plugins": [
          {
            "type": "flannel",
```

```
            "delegate": {
              "hairpinMode": true,
              "isDefaultGateway": true
            }
          },
          {
            "type": "portmap",
            "capabilities": {
              "portMappings": true
            }
          }
        ]
      }
    net-conf.json: |
      {
        "Network": "10.244.0.0/16",
        "EnableNFTables": false,
        "Backend": {
          "Type": "vxlan"
        }
      }
  kind: ConfigMap
  metadata:
    labels:
      app: flannel
      k8s-app: flannel
      tier: node
    name: kube-flannel-cfg
    namespace: kube-flannel
  ---
  apiVersion: apps/v1
  kind: DaemonSet
  metadata:
    labels:
      app: flannel
      k8s-app: flannel
      tier: node
    name: kube-flannel-ds
    namespace: kube-flannel
  spec:
    selector:
      matchLabels:
        app: flannel
        k8s-app: flannel
    template:
      metadata:
        labels:
```

```
        app: flannel
        k8s-app: flannel
        tier: node
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/os
                operator: In
                values:
                - linux
      containers:
      - args:
        - --ip-masq
        - --kube-subnet-mgr
        command:
        - flanneld
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: EVENT_QUEUE_DEPTH
          value: "5000"
        # имя и тег распакованного в пп. 15.2 образа
        # flannel-flanneld:
        image: 10.0.2.20:5000/flannel-flanneld:<тег>
        name: kube-flannel
        resources:
          requests:
            cpu: 100m
            memory: 50Mi
        securityContext:
          capabilities:
            add:
            - NET_ADMIN
            - NET_RAW
          privileged: false
        volumeMounts:
        - mountPath: /run/flannel
          name: run
        - mountPath: /etc/kube-flannel/
```

```
            name: flannel-cfg
          - mountPath: /run/xtables.lock
            name: xtables-lock
        hostNetwork: true
        initContainers:
        - args:
          - -f
          - /flannel
          - /usr/libexec/cni/flannel
          command:
          - cp
          # имя и тег распакованного в пп. 15.2 образа
          # flannel-cni-plugin-flannel:
          image: 10.0.2.20:5000/flannel-cni-plugin-flannel:<тег>
          name: install-cni-plugin
          volumeMounts:
          - mountPath: /usr/libexec/cni
            name: cni-plugin
        - args:
          - -f
          - /etc/kube-flannel/cni-conf.json
          - /etc/cni/net.d/10-flannel.conflist
          command:
          - cp
          # имя и тег распакованного в пп. 15.2 образа
          # flannel-flanneld:
          image: 10.0.2.20:5000/flannel-flanneld:<тег>
          name: install-cni
          volumeMounts:
          - mountPath: /etc/cni/net.d
            name: cni
          - mountPath: /etc/kube-flannel/
            name: flannel-cfg
        priorityClassName: system-node-critical
        serviceAccountName: flannel
        tolerations:
        - effect: NoSchedule
          operator: Exists
        volumes:
        - hostPath:
            path: /run/flannel
          name: run
        - hostPath:
            path: /usr/libexec/cni
          name: cni-plugin
        - hostPath:
            path: /etc/cni/net.d
          name: cni
```

```
        - configMap:
            name: kube-flannel-cfg
          name: flannel-cfg
        - hostPath:
            path: /run/xtables.lock
            type: FileOrCreate
          name: xtables-lock
```

Сохраните внесенные в конфигурацию изменения.

15.5 На `master`-ноде примените внутреннюю конфигурацию сети в кластере:

```
kubectl apply -f kube-flannel.yaml
```

Потребуется некоторое время для настройки (от 2 до 5 минут).

16. После применения конфигурации проверьте список и статус системных компонентов Kubernetes, а также компонентов сетевого плагина `flannel`:

```
kubectl get all -n kube-system
kubectl get all -n kube-flannel
```

17. Выполните дополнительную настройку сети:

```
sudo hostnamectl set-hostname masternode
```

Проверьте изменения:

```
hostname && hostname -I
```

```
masternode
10.0.2.16
```

18. В файл `/etc/hosts` внесите данные о `master` и `worker`:

```
sudo nano /etc/hosts
```

```
10.0.2.16 masternode
10.0.2.17 worker
```

Для вывода команды присоединения `worker` к кластеру выполните:

```
kubeadm token create --print-join-command
```

```
kubeadm join 10.0.2.16:6443 --token bh5nw3.uz8a96nhao7u0ue2 \
--discovery-token-ca-cert-hash sha256:e3623f73dadb313ea38e5f-
04736da6b662925aa78b07d00c50fca6aa0f30e391
```

3.8 Настройка рабочих нод

1. Для работы `kubelet` отключите `swap`:

- выполните команду для разового отключения:

```
swapoff -a
```

- чтобы `swap` не появился после перезагрузки сервера, выполните команду:

```
swapoff -a && sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

2. Создайте файл для автозагрузки модулей ядра, необходимых для работы сервиса `containerd`:

```
nano /etc/modules-load.d/containerd.conf
```

```
overlay
br_netfilter
```

Загрузите модули в ядро:

```
modprobe overlay
modprobe br_netfilter
```

Проверьте, что данные модули работают:

```
lsmod | egrep "br_netfilter|overlay"
```

3. Создайте конфигурационный файл для работы сети внутри `kubernetes`:

```
nano /etc/sysctl.d/99-kubernetes-cri.conf
```

```
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
```

Примените параметры командой:

```
sysctl --system
```

4. Установите необходимые пакеты:

```
dnf install kubernetes kubernetes-kubeadm cri-tools iproute-tc \
ebtables socat conntrack git curl wget runc containerd
```

Важно! Если настраиваемый хост не имеет доступа в Интернет, потребуется предварительно настроить локальный репозиторий ОС, из которого будет производиться дальнейшая установка необходимых для работы пакетов.

5. Настройте проброс портов в `iptables`:

```
iptables -P FORWARD ACCEPT
```

6. Запустите службу `containerd` и добавьте ее в автозагрузку:

```
systemctl enable --now containerd
```

7. Установите настройки по умолчанию для конфигурации контейнера:

```
containerd config default | sudo tee /etc/containerd/config.toml
```

8. Внесите изменения в файл конфигурации `/etc/containerd/config.toml`:

```
nano /etc/containerd/config.toml
```

8.1 Настройте доступ к локальному `registry`.
Для этого замените строку:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors]
```

на:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."10.0.2.20\
:5000"]
    endpoint = ["http://10.0.2.20:5000"]
```

8.2 Настройте доступ к контейнеру `pause`. Обратите внимание, что текущая версия Kubernetes совместима с контейнером `pause` только версии `3.7`.
Замените строку:

```
[plugins."io.containerd.grpc.v1.cri"]
    sandbox_image = "registry.k8s.io/pause:3.8"
```

на:

```
[plugins."io.containerd.grpc.v1.cri"]
    sandbox_image = "10.0.2.20:5000/pause:3.7"
```

8.3 Настройте расположение сетевых плагинов.
Для этого замените строку:

```
[plugins."io.containerd.grpc.v1.cri".cni]
    bin_dir = "/opt/cni/bin"
```

на:

```
[plugins."io.containerd.grpc.v1.cri".cni]
    bin_dir = "/usr/libexec/cni"
```

8.4 Настройте использование драйвера `cgroup`.
Для этого замените строку:

```
  [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.
options]
  ...
  SystemdCgroup = false
```

на:

```
  [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.
options]
  ...
  SystemdCgroup = true
```

Сохраните внесенные изменения.
9. Перезагрузите конфигурации служб:

```
  systemctl daemon-reload
```

10. Перезапустите службу `containerd`:

```
  systemctl restart containerd
```

11. Добавьте службу `kubelet` в автозагрузку:

```
  systemctl enable kubelet.service
```

12. Выполните дополнительную настройку сети:

```
  sudo hostnamectl set-hostname worker
```

Проверьте изменения:

```
  hostname && hostname -I
```

```
  worker
  10.0.2.17
```

13. В файл `/etc/hosts` внесите данные о `master` и `worker`:

```
  sudo nano /etc/hosts
```

```
  10.0.2.16 masternode
  10.0.2.17 worker
```

После этого можно приступать к присоединению `worker` к кластеру.

3.9 Подключение рабочей ноды к кластеру

1. На master-ноде получите команду присоединения worker:

```
kubeadm token create --print-join-command
```

```
kubeadm join 10.0.2.16:6443 --token bh5nw3.uz8a96nhao7u0ue2 \
   --discovery-token-ca-cert-hash sha256:e3623f73dadb313ea38e5f\
04736da6b662925aa78b07d00c50fca6aa0f30e391
```

2. На worker выполните полученную команду:

```
kubeadm join 10.0.2.16:6443 --token bh5nw3.uz8a96nhao7u0ue2 \
--discovery-token-ca-cert-hash sha256:e3623f73dadb313ea38e5f\
04736da6b662925aa78b07d00c50fca6aa0f30e391
```

```
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl
-n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/
kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to
file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS
Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a
response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node
join the cluster.
```

3. На master выполните команду для вывода списка нод кластера и их статуса:

```
kubectl get nodes
```

```
NAME                    STATUS   ROLES           AGE    VERSION
localhost.localdomain   Ready    control-plane   5h3m   v1.24.17
worker                  Ready    <none>          89s    v1.24.17
```

3.10 Безопасность в среде контейнеризации Kubernetes

3.10.1 Изоляция контейнеров

В среде контейнеризации обеспечиваются следующие механизмы изоляции контейнеров:

- изоляция пространств идентификаторов процессов контейнеров (PID);
- изоляция пространств имен для межпроцессного взаимодействия контейнеров (IPC);
- изоляция пространств имен для пользователей и групп контейнеров (USER);
- изоляция пространств имен хостов и доменов контейнеров (UTS);
- изоляция сетевых пространств имен контейнеров (NETWORK);
- изоляция пространств имен для иерархии каталогов контейнеров (MOUNT).

Изоляция контейнеров в Kubernetes с использованием containerd реализуется на нескольких уровнях:

1. Пространства имен (namespaces) – каждый контейнер работает в своем собственном пространстве имен, что позволяет изолировать процессы, сети и файловые системы. Kubernetes использует различные типы namespaces, такие как PID, Network, Mount и другие.

2. Контрольные группы (cgroups) – контейнеры используют control groups (cgroups) для ограничения и мониторинга использования ресурсов (ЦП, память, диск и т. д.), что позволяет управлять ресурсами на уровне контейнера и предотвращает их чрезмерное использование.

3. Сетевые политики (network policies) – в Kubernetes доступна настройка сетевых политик для управления трафиком между подами. Это позволяет ограничивать доступ к определенным сервисам и ресурсам.

4. Монтирование томов (volumes) – Kubernetes использует механизмы монтирования томов, чтобы обеспечить изоляцию данных между контейнерами. Каждый контейнер может иметь свои собственные тома, что предотвращает нежелательный доступ к данным других контейнеров.

3.10.2 Выявление уязвимостей в образах контейнеров

Для выявления уязвимостей в образах контейнеров совместно со встроенными средствами защиты ОС применяется сертифицированное программное обеспечение для анализа защищённости и управления ИБ – RedCheck. Подробную информацию см. в п. 2.6.2 «Выявление уязвимостей в образах контейнеров».

Примечание. Рекомендуется устанавливать RedCheck непосредственно **на узлы кластера** Kubernetes.

3.10.3 Проверка корректности конфигурации контейнеров

По умолчанию в контейнере недоступны устройства хостовой системы.

Для получения списка устройств контейнера необходимо запустить контейнер командой вида:

```
kubectl run <имя_контейнера> --image=<путь_к_образу>:<тег>
```

После запуска контейнера выполните вход в виртуальную консоль:

```
kubectl exec -it <имя_контейнера> -- /bin/bash
```

Далее необходимо проверить список доступных контейнеру устройств:

```
ls /dev
```

```
core  full    null  pts     shm     stdin   termination-log
urandom
fd    mqueue  ptmx  random  stderr  stdout  tty
zero
```

Для отображения информации о шинах PCI в контейнере и подключенных к ним устройствах выполните:

```
lspci
```

```
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC
[Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/
Triton II]
00:01.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE
(rev 01)
00:02.0 VGA compatible controller: VMware SVGA II Adapter
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit
Ethernet Controller (rev 02)
00:04.0 System peripheral: InnoTek Systemberatung GmbH VirtualBox
Guest Service
00:05.0 Multimedia audio controller: Intel Corporation 82801AA
AC'97 Audio Controller (rev 01)
00:06.0 USB controller: Apple Inc. KeyLargo/Intrepid USB
00:07.0 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev
08)
```

Для отображения информации о USB-шинах в контейнере и подключенных к ним устройствах выполните:

```
lsusb
```

```
Bus 001 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 001 Device 001: ID 1d6b:0001
```

Для отображения информации о подключенных дисках в контейнере используются команды `fdisk` (для жестких дисков) и `blkid` (для блочных устройств).

Например:

```
fdisk -l
```

Однако в рассматриваемом контейнере вывод указанных команд должен отсут-

ствовать, т. к. при запуске контейнера к ним не подключались диски или хранилища.

Для получения списка доступных устройств на хостовой ОС выполните те же команды:

```
ls /dev
lspsi
lsusb
fdisk -l
```

Для ограничения прав прикладного программного обеспечения, выполняемого внутри контейнера, на использование вычислительных ресурсов хостовой ОС используются специальные файлы (с расширением `JSON` или `YAML`), называемые манифестами.

Пример содержимого файла манифеста для установки ограничения на использование ОЗУ:

```
apiVersion: v1
kind: Pod
metadata:
  name: <имя_пода>
spec:
  containers:
  - name: app
    image: <путь_к_образу>
    resources:
      requests:
        memory: "64Mi"  # гарантированный (минимальный) объем ОЗУ
      limits:
        memory: "128Mi"  # максимальный объем ОЗУ
```

При попытке запуска контейнера с максимально выделенным объемом ОЗУ в `128 МБ` будет выведено сообщение об ошибке вида:

```
Failed to create pod sandbox: rpc error: code = Unknown desc =
failed to create containerd task: failed to create shim task: OCI
runtime create failed: runc create failed: unable to start container
process: container init was OOM-killed (memory limit too low?):
unknown
```

Для ограничения в контейнере пропускной способности для операций ввода-вывода необходимо перед запуском контейнера задать ограничения в файле конфигурации:

```
sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' \
/etc/containerd/config.toml
echo "253:0 riops=10 wiops=10" > /sys/fs/cgroup/kubepods.slice/
io.max
```

где `253:0` – это диск, который использует kubernetes (проверка осуществляется командой `cat /proc/partitions`).

После этого запустить контейнер и проверить статистику командой:

```
time dd if=/dev/random of=/tmp/file1 bs=1M count=1024 oflag=direct
```

```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 102.33 s, 10.5 MB/s

real 1m42.335s
user 0m0.015s
sys 0m3.329s
```

При запуске контейнера без ограничения количества операций ввода/вывода статистика выглядит следующим образом:

```
time dd if=/dev/random of=/tmp/file1 bs=1M count=1024 oflag=direct
```

```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 4.40604 s, 244 MB/s

real 0m4.411s
user 0m0.005s
sys 0m3.020s
```

Для монтирования корневой файловой системы хостовой операционной системы в режиме "только для чтения" тоже используются манифесты.

Пример содержимого файла манифеста для монтирования корневой файловой системы хостовой ОС:

```
apiVersion: v1
kind: Pod
metadata:
  name: <имя_пода>
spec:
  containers:
  - name: <имя_контейнера>
    image: <путь_к_образу>
    volumeMounts:
    - name: host-volume
      mountPath: /mnt/host
      readOnly: true

  volumes:
  - name: host-volume
    hostPath:
      path: /root    # каталог монтирования
      type: Directory
```

Для вывода содержимого каталога контейнера /mnt/host/ выполните:

```
ls mnt/host/
```

```
anaconda-ks.cfg run
```

При попытке создать какой-либо файл в данном каталоге будет выведено сообщение об ошибке вида:

```
touch mnt/host/d
```

```
touch: cannot touch 'mnt/host/d': Read-only file system
```

3.10.4 Контроль целостности контейнеров и их образов

Контроль целостности образов контейнеров и их исполняемых файлов производится инструментом хостовой ОС — IMA, контроль целостности сведений о событиях безопасности осуществляется средством верификации целостности хостовой ОС — afick. Подробную информацию об инструментах см. в подразделе «Afick — верификация целостности» и подразделе «Контроль целостности запускаемых компонентов программного обеспечения» Руководства администратора. Часть 1.

По умолчанию из-за высокой нагрузки на ЭВМ функция контроля целостности выключена. Включение производится в файле настроек /etc/containerd/config.toml установкой значения параметра validate в true в секции [plugins."io.containerd.grpc.v1.cri"]. Проверка целостности по расписанию (1 раз в неделю) по умолчанию также отключена по тем же причинам.

Для включения проверки целостности образов контейнеров и файлов конфигурации необходимо выполнить:

```
systemctl enable containerd-validate.timer --now
```

Для изменения настроек расписания проверки целостности образов контейнеров и файлов конфигурации средства контейнеризации необходимо изменить настройки таймера в файле /usr/lib/systemd/system/containerd-validate.timer в секции [Timer].

По умолчанию файл имеет следующее содержимое:

```
[Unit]
Description=Checking the integrity of container images and con-
figuration file once a week
ConditionVirtualization=!container
ConditionPathExists=/var/run/containerd/containerd.sock
ConditionFileIsExecutable=/usr/bin/ctr

[Timer]
OnCalendar=weekly
AccuracySec=1h
Persistent=true
```

```
[Install]
WantedBy=timers.target
```

Примечание. Не рекомендуется осуществлять проверку контроля целостности образов контейнеров и файла конфигурации средства контейнеризации чаще 1 раза в час. Индивидуальные настройки зависят от количества установленных образов и вычислительной мощности отдельно взятой ЭВМ.

После внесения каких-либо изменений в файл настройки таймера необходимо его перезапустить, чтобы внесенные изменения вступили в силу:

```
systemctl restart containerd-validate.timer
```

Отключение проверки целостности образов по расписанию реализуется выполнением команды:

```
systemctl disable containerd-validate.timer --now
```

При попытке инициализировать кластер, целостность образов контейнеров которого нарушена, будет выведено сообщение об ошибке примерно следующего вида (соответственно запуск (создание) контейнера будет невозможным):

```
Unfortunately, an error has occurred:
  timed out waiting for the condition

This error is likely caused by:
    - The kubelet is not running
    - The kubelet is unhealthy due to a misconfiguration of the
node in some way (required cgroups disabled)

If you are on a systemd-powered system, you can try to troubleshoot
the error with the following commands:
    - 'systemctl status kubelet'
    - 'journalctl -xeu kubelet'

Additionally, a control plane component may have crashed or exited
when started by the container runtime.
To troubleshoot, list all containers using your preferred container
runtimes CLI.
Here is one example how you may list all running Kubernetes containers
by using crictl:
    - 'crictl --runtime-endpoint unix:///var/run/containerd/con-
tainerd.sock ps -a | grep kube | grep -v pause'
    Once you have found the failing container, you can inspect its
logs with:
    - 'crictl --runtime-endpoint unix:///var/run/containerd/con-
tainerd.sock logs CONTAINERID'
error execution phase wait-control-plane: couldn't initialize a
Kubernetes cluster
```

```
To see the stack trace of this error execute with --v=5 or higher
```

В журнале хостовой ОС будет зафиксирована попытка запуска контейнера из поврежденного образа. Запись в журнале будет иметь примерно следующий вид:

```
journalctl -xe -u containerd
```

```
июн 22 11:39:17 master containerd[7376]: time="2025-06-22T11:39:
17.419449297+03:00" level=error msg="the image is corupted"
id=dab44fd3-c380-43e6-904b-ec08a69befcf image="sha256:dd75ec974b0
a2a6f6bb47001ba09207976e625db898d1b16735528c009cb171c" result=error
type="Validate image" user=
```

либо:

```
grep "the image is corupted" /var/log/containerd/containerd.log
```

```
2025-06-22T11:52:01.419977+03:00 master containerd[7376]:
time="2025-06-22T11:52:01.419806259+03:00" level=error msg="the
image is corupted" id=1d7484ad-23eb-49af-babb-c6d65f7fab46
image="sha256:dd75ec974b0a2a6f6bb47001ba09207976e625db898d1b1673552
8c009cb171c" result=error type="Validate image" user=
```

Также средством контейнеризации осуществляется контроль целостности параметров настройки с помощью инструмента хостовой ОС — IMA.

В журнале безопасности хостовой ОС будут зарегистрированы соответствующие записи о нарушении целостности файла настроек средства контейнеризации:

```
journalctl -xe -u containerd
```

```
июн 22 13:53:36 master containerd[17923]: time="2025-06-22T13:53:
36+03:00" level=warning msg="The configuration file did not pass
integrity control!" id=c050ddeb-6646-48b5-8b8a-501523bec452
result=OK type="Vadlidate config file" user=
```

3.10.5 Регистрация событий безопасности

Для каждой функции безопасности в Kubernetes определен перечень событий, необходимых для регистрации и учета.

Регистрация событий безопасности производится средствами хостовой ОС.

В журнале безопасности регистрируются такие виды событий, как:

- создание и удаление образов контейнеров (модификация контейнеров не поддерживается);
- получение доступа к образам контейнеров;
- загрузка, запуск, остановка и удаление контейнеров с указанием причины остановки;
- факты нарушения целостности объектов контроля.

Просмотр зарегистрированных действий осуществляется с помощью команды (с правами суперпользователя):

```
journalctl -xe -u containerd
```

События по загрузке образов, запуску и остановке контейнеров отображаются в виде следующих записей:

```
   июн 22 15:52:48 master containerd[32463]: time="2025-06-22T12:29:
18.122244013+03:00" level=info msg="Pulled image \"registry.k8s.io/
pause:3.7\" with image id \"sha256:221177c6082a88ea4f6240ab2450d540
955ac6f4d5454f0e15751b653ebda165\", repo tag \"registry.k8s.io/
pause:3.7\", repo digest \"registry.k8s.io/pause@sha256:bb6ed397957
e9ca7c65ada0db5c5d1c707c9c8afc80a94acbe69f3ae76988f0c\", size \"311
278\" in 2.900431018s" id=f2b1b913-aa15-478c-ae34-c8328b5bf0ad
image="registry.k8s.io/pause:3.7" result=OK type="Pull the image"
user=
   ...
   июн 22 15:52:48 master containerd[32463]: time="2025-06-22T12:45:
26.334294907+03:00" level=info msg="Container \"6e082158d26eadd7b5
aa16b00cd62820dafc1d7fe77e5794bb00804b8f34ec7a\" was created"
container=6e082158d26eadd7b5aa16b00cd62820dafc1d7fe77e5794bb00804b
8f34ec7a id=932f48b6-55c3-4bc3-b1ce-92bdded71495 result=OK
type="Create container" user=
   ...
   июн 22 16:02:57 master containerd[3453]: time="2025-06-22T11:36:
08.225898414+03:00" level=info msg="StopPodSandbox for \"8553881080
6acb4f8ccf74c561255225ad202657255823d7f9fc75e6be0327cf\""
   июн 22 16:02:57 master containerd[3453]: time="2025-06-22T11:36:
08.225898414+03:00" level=info msg="Container to stop \"85538810806
acb4f8ccf74c561255225ad202657255823d7f9fc75e6be0327cf\" must be in
running or unknown state, current state \"CONTAINER_EXITED\""
container=aa44eb6e8a5d7519aaa8c19132f935172c176fdfc792f2fc114fade48
a5d0323 id=ec3bcc1e-f074-4d92-b190-f01d390de474 result=OK type="Stop the
container" user=
```

Журнал событий безопасности средства контейнеризации доступен только для чтения. Проверку установленных прав на журнал можно произвести командой:

```
ls -l /var/log/containerd/containerd.log
```

```
   -rw-r--r-- 1 root root 155414 июн 22 16:03 /var/log/containerd/
containerd.log
```

Стандартная настройка ротации и архивации журнала производится при достижении размера `10 МБ`. Настройка параметров осуществляется только суперпользователем `root` в файле `/etc/logrotate.d/containerd`.

Содержимое файла имеет следующий вид:

```
/var/log/containerd/containerd.log {
    missingok
    maxsize 10M
    compress
    rotate 10
    copytruncate
    delaycompress
    create 0600 root root
}
```

При достижении файлом журнала максимально установленного размера будет выполнено его архивирование. Копия журнала будет сохранена в каталоге `/var/log/containerd/` с указанием даты ее создания.

Пример файла:

```
ls -l /var/log/containerd/
```

```
-rw-r--r--. 1 root root  3196 июн 23 12:38 containerd.log
-rw-r--r--. 1 root root 13497 июн 23 14:15 containerd.log-20250623
```

4. Сканер уязвимостей Trivy

4.1 Общие сведения

Trivy – универсальный сканер безопасности. Trivy позволяет обнаруживать уязвимости, ошибки конфигураций, открытые конфиденциальные данные (секреты), файлы лицензий в различных программных компонентах.

Сканирование компонентов на уязвимости производится согласно следующим базам уязвимостей:

- база уязвимостей `trivy`;
- база уязвимостей РЕД ОС, публикуемая на сайте РЕД ОС в виде `xml`-файла формата `OVAL` – `https://redos.red-soft.ru/support/secure/redos.xml`.
- база уязвимостей БДУ ФСТЭК России, публикуемая на сайте ФСТЭК России – `https://bdu.fstec.ru/vul`.

4.2 Установка

Для установки сканера `trivy` выполните команду:

```
dnf install trivy trivy-check trivy-rsyslog trivy-logrotate
```

После установки пакетов необходимо обновить конфигурацию системных служб и перезапустить службы `rsyslog` и `logrotate`:

```
systemctl daemon-reload
systemctl restart rsyslog
systemctl restart logrotate
```

Примечание. Дальнейшая работа со сканером `trivy` должна выполняться от имени пользователя, **обладающего необходимыми разрешениями** на взаимодействие с объектом сканирования.

Для удобства администраторов были разработаны скрипты `trivy-check-image` и `trivy-check-k8s`, которые позволяют выполнять сканирование `docker`-образов и кластера Kubernetes на наличие уязвимостей и ошибок конфигураций с выводом краткой информации о результатах сканирования и сохранением сведений в журнале безопасности.

Синтаксис утилиты `trivy` имеет вид:

```
trivy [<опции>] <команда> [<опции>] <объект_сканирования>
trivy [<команда>]
```

где:
- `<команда>` – позволяет задать настройки утилиты, а также определяет выполняемые действия над объектом сканирования;
- `<объект_сканирования>` – задает объект сканирования (образ, каталог, репозиторий, файловая система).

Основные команды утилиты:
- `clean` – очистка кешированных файлов;
- `completion` – генерация скрипта автозаполнения для указанной оболочки;
- `convert` – конвертация отчета `JSON` в другой формат;
- `registry` – управление проверкой подлинности реестра;
- `server` – переключение в режим сервера;
- `help` – вывод справки по использованию утилиты;
- `version` – вывод версии утилиты.

Основные команды сканирования:
- `config` – запуск сканирования файлов конфигурации на наличие ошибок;
- `filesystem` (`fs`) – запуск сканирования локальной файловой системы;
- `image` – запуск сканирования образа контейнера;
- `kubernetes` (`k8s`) – запуск сканирования кластера Kubernetes;
- `repository` – запуск сканирования репозитория;
- `rootfs` – запуск сканирования `rootfs`;
- `sbom` – запуск сканирования компонентов ОС и их зависимостей (SBOM);
- `vm` – запуск сканирования образа виртуальной машины.

Команды управления:
- `module` – управление модулями;
- `plugin` – управление плагинами.

Опции утилиты:
- `--cache-dir <путь_каталогу>` – путь к каталогу кеша утилиты (по умолчанию `/root/.cache/trivy`);
- `-c`, `--config <файл_конфигурации>` – путь к файлу конфигурации утилиты (по умолчанию `trivy.yaml`);
- `-d`, `--debug` – режим отладки;
- `-f`, `--format <формат>` – формат (по умолчанию `json`);
- `--generate-default-config` – запись конфигурации по умолчанию в файл `trivy-default.yaml`;
- `-h`, `--help` – вывод справки по утилите;
- `--insecure` – разрешение небезопасных подключений к серверу;
- `-q`, `--quiet` – подавление индикатора выполнения и вывода журнала;
- `--timeout <время>` – таймаут (по умолчанию 5 мин.);
- `--using-bdu` – использование базы уязвимостей БДУ ФСТЭК России;

- `-v`, `--version` – вывод версии утилиты.

По умолчанию `trivy` выполняет сканирование компонентов ОС, используя только собственную базу уязвимостей и базу уязвимостей РЕД ОС. Для включения поддержки сканирования компонентов ОС с использованием базы уязвимостей ФСТЭК России необходимо дополнительно использовать опцию `--using-bdu`.

Для вывода информации по использованию утилиты `trivy` выполните:

```
trivy --help
```

Для вывода подробной информации по использованию какой-либо команды выполните:

```
trivy <команда> --help
```

Синтаксис утилиты `trivy-check` имеет вид:

```
trivy-check-image [<опции>] <объект_сканирования>
trivy-check-k8s [<опции>]
```

где `<объект_сканирования>` – задает объект сканирования (образ, кластер).

Утилита `trivy-check` имеет одну опцию – `-b`, при использовании которой сканирование проблем безопасности выполняется по БДУ ФСТЭК России. Если опция `-b` не указана, сканирование производится по базам уязвимостей `trivy` и РЕД ОС.

Базы уязвимостей загружаются при первом запуске сканирования и хранятся в локальном кеше утилиты в каталоге `/root/.cache/trivy`. Кеш с загруженными базами уязвимостей обновляется 2 раза в день.

В случае возникновения проблем со сканированием компонентов ОС с использованием базы уязвимостей ФСТЭК России см. подраздел 4.6 «Очистка кеша утилиты».

4.3 Сканирование локальной файловой системы

Trivy позволяет сканировать локальную файловую систему на выявление следующих проблем:

- уязвимости;
- ошибки конфигурации;
- конфиденциальные данные (секреты);
- лицензии.

Для сканирования локальной файловой системы используется команда вида:

```
trivy filesystem [<опции>] <путь_к_объекту_сканирования>
```

По умолчанию включено сканирование на уязвимости (`vuln`). Настроить сканирование на наличие других проблем можно с помощью параметра `--scaners`. Параметр `--scanners` может принимать следующие значения (несколько значений указывается через запятую):

- `vuln` – уязвимости;
- `misconfig` – ошибки конфигурации;

- `secret` – раскрытые конфиденциальные данные;
- `license` – корректность файлов лицензий.

Пример сканирования локальной файловой системы на корректность файлов лицензий и запись отчета в файл:

```
trivy fs --scanners license --report summary / -o ./report-lic
```

Для просмотра файла отчета необходимо выполнить:

```
cat ./report-lic
```

Сформированный файл отчета (часть файла отчета) с отображением найденных проблем представлен на рисунке 4.1.

```
Report Summary

┌──────────────┬──────┬──────────┐
│    Target    │ Type │ Licenses │
├──────────────┼──────┼──────────┤
│ OS Packages  │  -   │   2056   │
└──────────────┴──────┴──────────┘
Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)


OS Packages (license)
=====================
Total: 2056 (UNKNOWN: 803, LOW: 369, MEDIUM: 24, HIGH: 858, CRITICAL: 2)

┌──────────────────────────────┬──────────────────────────────────────────────────┬────────────────┬──────────┐
│           Package            │                     License                      │ Classification │ Severity │
├──────────────────────────────┼──────────────────────────────────────────────────┼────────────────┼──────────┤
│ 7zip                         │ BSD-3-Clause AND LGPL-2.1-or-later AND SUSE-Public-Domain │ unknown        │ UNKNOWN  │
├──────────────────────────────┼──────────────────────────────────────────────────┼────────────────┼──────────┤
│ Box2D                        │ zlib                                             │ notice         │ LOW      │
├──────────────────────────────┼──────────────────────────────────────────────────┤                │          │
│ ImageMagick                  │ ImageMagick                                      │                │          │
├──────────────────────────────┤                                                  │                │          │
│ ImageMagick-libs             │                                                  │                │          │
├──────────────────────────────┼──────────────────────────────────────────────────┼────────────────┼──────────┤
│ LibRaw                       │ BSD and (CDDL-1.0 or LGPLv2)                     │ unknown        │ UNKNOWN  │
├──────────────────────────────┼──────────────────────────────────────────────────┼────────────────┼──────────┤
│ ModemManager                 │ GPL-2.0-or-later                                 │ restricted     │ HIGH     │
├──────────────────────────────┼──────────────────────────────────────────────────┤                │          │
│ ModemManager-glib            │ LGPL-2.1-or-later                                │                │          │
├──────────────────────────────┼──────────────────────────────────────────────────┼────────────────┼──────────┤
│ NetworkManager               │ GPLv2+ and LGPLv2+                               │ unknown        │ UNKNOWN  │
├──────────────────────────────┤                                                  │                │          │
│ NetworkManager-adsl          │                                                  │                │          │
├──────────────────────────────┤                                                  │                │          │
│ NetworkManager-bluetooth     │                                                  │                │          │
├──────────────────────────────┼──────────────────────────────────────────────────┼────────────────┼──────────┤
│ NetworkManager-iodine        │ GPLv2+                                           │ restricted     │ HIGH     │
├──────────────────────────────┤                                                  │                │          │
│ NetworkManager-iodine-gnome  │                                                  │                │          │
├──────────────────────────────┤                                                  │                │          │
│ NetworkManager-l2tp          │                                                  │                │          │
├──────────────────────────────┤                                                  │                │          │
│ NetworkManager-l2tp-gnome    │                                                  │                │          │
```

Рисунок 4.1 — Сформированный файл отчета сканирования локальной ФС

Пример сканирования локальной файловой системы на наличие раскрытых конфиденциальных данных:

```
trivy fs --scanners secret --report summary /etc/ssh/
```

Сформированный отчет с отображением найденных проблем представлен на рисунке 4.2.

Рисунок 4.2 — Сформированный отчет сканирования локальной ФС

Подробную информацию по использованию команды см. в справке:

```
trivy filesystem --help
```

4.4 Сканирование образов контейнеров

Trivy поддерживает два типа сканируемых объектов при проверке образов контейнеров:

- файлы внутри образов контейнеров;
- метаданные образов контейнеров.

4.4.1 Сканирование файлов внутри образа контейнера

Файлы внутри образов контейнеров – это файлы и директории, которые находятся в файловой системе контейнера. Они могут включать исполняемые файлы, библиотеки, конфигурационные файлы и другие ресурсы, необходимые для работы приложения. При сканировании Trivy анализирует эти файлы на наличие уязвимостей, которые могут быть связаны с установленными пакетами, библиотеками и программами.

Для сканирования файлов образа контейнера и вывода подробного отчета о найденных проблемах используется команда вида:

```
trivy image [<опции>] <имя_контейнера>:<тег>
```

Примечание. Сканирование образов контейнеров должно выполняться от имени пользователя, **обладающего необходимыми разрешениями** на взаимодействие с объектом сканирования.

Trivy позволяет сканировать файлы внутри образов контейнеров на наличие следующих проблем безопасности:

- уязвимости;
- ошибки конфигураций;
- раскрытые конфиденциальные данные;
- лицензии.

По умолчанию включено сканирование на уязвимости (`vuln`) и раскрытые конфиденциальные данные (`secret`). Настроить сканирование на наличие других проблем можно с помощью параметра `--scaners`. Параметр `--scanners` может принимать следующие значения (несколько значений указывается через запятую):

- `vuln` – уязвимости;
- `misconfig` – ошибки конфигурации;
- `secret` – раскрытые конфиденциальные данные;
- `license` – корректность файлов лицензий.

Для сканирования `docker`-образа на уязвимости по базам уязвимостей `trivy` и РЕД ОС, а также на корректность файлов конфигурации и регистрации события в журнале безопасности используется команда вида:

```
trivy-check-image <имя_контейнера>:<тег>
```

Для сканирования `docker`-образа на уязвимости по базам уязвимостей `trivy`, РЕД ОС и ФСТЭК России, а также на корректность файлов конфигурации и регистрации события в журнале безопасности используется команда вида:

```
trivy-check-image -b <имя_контейнера>:<тег>
```

Важно! Для получения более детальной информации о результатах проверки образов контейнеров на наличие проблем безопасности рекомендуется после выполнения команды `trivy-check-image`, в случае обнаружения проблем, дополнительно выполнить команду `image` утилиты `trivy`. ■

Пример сканирования образа контейнера на наличие проблем безопасности с выводом подробного отчета в терминал и записью события в журнал безопасности:

```
trivy-check-image -b registry.red-soft.ru/ubi7/firebird-3:latest
trivy image --using-bdu registry.red-soft.ru/ubi7/firebird-3:latest
```

Сформированный отчет представлен на рисунке 4.3.

```
Report Summary

| Target                                                  | Type  | Vulnerabilities |
|---|---|---|
| registry.red-soft.ru/ubi7/firebird-3:latest (redos 7.3) | redos |        1        |

Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)

registry.red-soft.ru/ubi7/firebird-3:latest (redos 7.3)

Total: 1 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 1, CRITICAL: 0)

| Library | Vulnerability | Severity | Status | Installed Version | Fixed Version | Title |
|---|---|---|---|---|---|---|
| firebird | BDU:2024-02303 | HIGH | fixed | 3.0.7.33374-1.el7 | 4.0.4.3010-1 | Уязвимость системы управления базами данных Firebird, позволяющая нарушителю вызвать отказ в обслуживании... https://bdu.fstec.ru/vul/2024-02303 |
```

Рисунок 4.3 — Сформированный отчет сканирования файлов внутри образа контейнера

4.4.2 Сканирование метаданных образа контейнера

Метаданные – это информация о самом образе контейнера, такая как теги, имя, размер, дата создания, а также информация о слоях образа и их зависимостях. Trivy использует метаданные для определения контекста образа и его зависимостей. Метаданные могут помочь понять, какие пакеты были установлены и какие слои были добавлены, что важно для более точного анализа уязвимостей.

Каждый образ контейнера имеет собственную конфигурацию. Команды `docker inspect <имя_контейнера>:<тег>` и `docker history <имя_контейнера>:<тег>` отображают информацию в соответствии с конфигурацией.

Trivy позволяет сканировать конфигурацию контейнера на наличие следующих проблем безопасности:

- ошибки конфигураций;
- раскрытые конфиденциальные данные.

При сканировании на наличие ошибок конфигураций `trivy` преобразует конфигурацию контейнера в `dockerfile` и обрабатывает ее как `dockerfile`.

При сканировании на наличие раскрытых конфиденциальных данных `trivy` преобразует конфигурацию контейнера в формат `JSON` и выполняет проверку. Такой подход может быть полезен для переменных среды, которые могут случайно получить учетные данные.

По умолчанию сканирование метаданных контейнера отключено. Для его включения необходимо использовать параметр `--image-config-scanners`. Параметр `--image-config-scanners` может принимать следующие значения (несколько значений указывается через запятую):

- `misconfig` – ошибки конфигурации;
- `secret` – раскрытые конфиденциальные данные.

Примечание. Сканирование образов контейнеров должно выполняться от имени пользователя, **обладающего необходимыми разрешениями** на взаимодействие с объектом сканирования.

Пример сканирования метаданных образа контейнера на наличие ошибок конфигурации:

```
trivy image --image-config-scanners misconfig \
registry.red-soft.ru/ubi7/firebird-3:latest
```

Сформированный отчет с отображением найденных проблем представлен на рисунке 4.4.

```
Report Summary

| Target                                                | Type | Vulnerabilities |
|---|---|---|
| registry.red-soft.ru/ubi7/firebird-3:latest (redos 7.3) | redos | 1 |

Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)

registry.red-soft.ru/ubi7/firebird-3:latest (redos 7.3)

Total: 1 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 1, CRITICAL: 0)

| Library | Vulnerability | Severity | Status | Installed Version | Fixed Version | Title |
|---|---|---|---|---|---|---|
| firebird | ROS-20240522-07 | HIGH | fixed | 3.0.7.33374-1.el7 | 4.0.4.3010-1 | Уязвимость firebird https://redos.red-soft.ru/search/?q=ROS-20240522-07 |
```

Рисунок 4.4 — Сформированный отчет сканирования метаданных образа контейнера

4.5 Сканирование кластера Kubernetes

Trivy может подключиться к кластеру Kubernetes и сканировать его на наличие проблем безопасности с помощью команды вида:

```
trivy kubernetes [<опции>] <объект_сканирования>
```

Примечание. Сканирование кластера Kubernetes должно выполняться на `Master`-ноде от имени пользователя, **обладающего разрешениями** на управление кластером.

Указать проблемы безопасности, которые должны быть обнаружены, можно с помощью параметра `--scanners`. Параметр `--scanners` может принимать следующие значения (несколько значений указывается через запятую):

- `vuln` – уязвимости;
- `misconfig` – ошибки конфигурации;
- `secret` – раскрытые конфиденциальные данные;
- `rbac` – нарушения правил ролевого контроля доступа.

По умолчанию сканирование проводится на наличие каждой проблемы безопасности.

Подробную информацию по использованию команды см. в справке:

```
trivy kubernetes --help
```

Для сканирования кластера Kubernetes по базам уязвимостей `trivy` и РЕД ОС, а также на корректность файлов конфигурации и регистрации события в журнале безопасности используется команда вида:

```
trivy-check-k8s
```

Для сканирования кластера Kubernetes по базам уязвимостей `trivy`, РЕД ОС и ФСТЭК России, а также на корректность файлов конфигурации и регистрации события в журнале безопасности используется команда вида:

```
trivy-check-k8s -b
```

Важно! Для получения более детальной информации о результатах проверки кластера Kubernetes на наличие проблем безопасности рекомендуется после выполнения команды `trivy-check-k8s`, в случае обнаружения проблем, дополнительно выполнить команду `k8s` утилиты `trivy`.

Для корректной проверки кластера на наличие проблем безопасности необходимо последовательно выполнять команды `trivy-check-k8s` и `trivy k8s`. ■

Пример сканирования кластера на наличие уязвимостей по базам уязвимостей `trivy` и РЕД ОС с выводом краткого отчета и записью события в журнал безопасности:

```
trivy-check-k8s
trivy k8s --scanners vuln --report summary --disable-node-collector
```

Сформированный отчет с отображением найденных проблем представлен на рисунке 4.5.

```
2025-05-12T16:22:57+03:00       INFO    Scanning K8s... K8s="kubernetes-admin@kubernetes"
206 / 206 [-----------------------------------------------------------------------------------------

Summary Report for kubernetes-admin@kubernetes


Workload Assessment
```

Namespace	Resource	Vulnerabilities				
		C	H	M	L	U
kube-flannel	DaemonSet/kube-flannel-ds		2	2		2

```
Severities: C=CRITICAL H=HIGH M=MEDIUM L=LOW U=UNKNOWN


Infra Assessment
```

Namespace	Resource	Vulnerabilities				
		C	H	M	L	U
kube-system	DaemonSet/kube-proxy			2	14	
kube-system	Deployment/coredns					3

```
Severities: C=CRITICAL H=HIGH M=MEDIUM L=LOW U=UNKNOWN
```

Рисунок 4.5 — Сформированный отчет сканирования кластера Kubernetes

Пример сканирования кластера на наличие уязвимостей по базам уязвимостей `trivy`, РЕД ОС и ФСТЭК России с выводом краткого отчета и записью события в журнал безопасности:

```
   trivy-check-k8s -b
   trivy k8s --using-bdu --scanners vuln --report summary \
--disable-node-collector
```

Сформированный отчет с отображением найденных проблем представлен на рисунке 4.6.

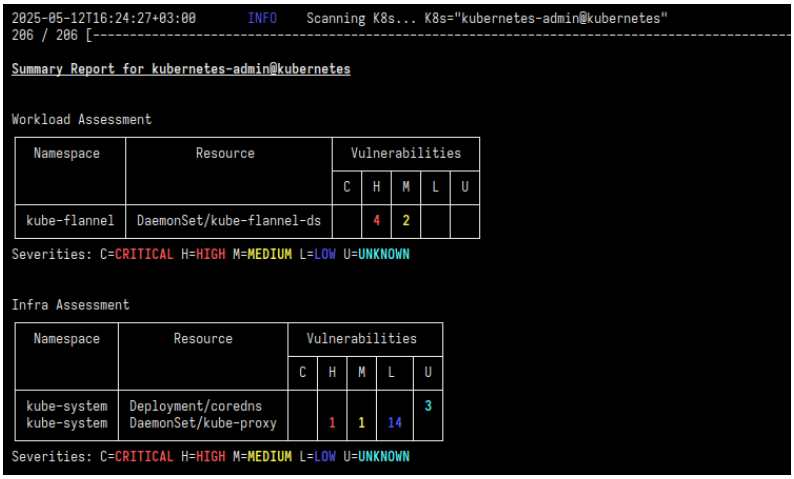

Рисунок 4.6 — Сформированный отчет сканирования кластера Kubernetes

Для вывода более полного отчета сканирования кластера на уязвимости используйте для ключа `report` значение `all`:

```
   trivy-check-k8s -b
   trivy k8s --using-bdu --scanners vuln --report all \
--disable-node-collector
```

Сформированный отчет (часть отчета) с отображением найденных проблем представлен на рисунке 4.7.

namespace: kube-flannel, daemonset: kube-flannel-ds

Total: 6 (UNKNOWN: 0, LOW: 0, MEDIUM: 2, HIGH: 4, CRITICAL: 0)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
libcrypto3	BDU:2025-01602	HIGH	fixed	3.3.2-r4	3.3.3-r0	Уязвимость режима SSL_VERIFY_PEER криптографической библиотеки OpenSSL, позволяющая нарушителю реализовать атаку типа «человек-посередине»... https://bdu.fstec.ru/vul/2025-01602
	BDU:2025-03280	MEDIUM			3.3.2-r5	Уязвимость библиотеки OpenSSL, связанная со скрытым временным каналом, позволяющая нарушителю восстановить закрытый... https://bdu.fstec.ru/vul/2025-03280
libssl3	BDU:2025-01602	HIGH			3.3.3-r0	Уязвимость режима SSL_VERIFY_PEER криптографической библиотеки OpenSSL, позволяющая нарушителю реализовать атаку типа «человек-посередине»... https://bdu.fstec.ru/vul/2025-01602
	BDU:2025-03280	MEDIUM			3.3.2-r5	Уязвимость библиотеки OpenSSL, связанная со скрытым временным каналом, позволяющая нарушителю восстановить закрытый... https://bdu.fstec.ru/vul/2025-03280
musl	BDU:2025-01869	HIGH		1.2.5-r9	1.2.5-r10	Уязвимость функции iconv библиотеки musl libc, позволяющая нарушителю выполнить произвольный код https://bdu.fstec.ru/vul/2025-01869
musl-utils						

Рисунок 4.7 — Сформированный отчет сканирования кластера Kubernetes

4.6 Очистка кеша утилиты

В случае, когда первое сканирование безопасности компонентов ОС запускается без использования БДУ ФСТЭК России (без опции `--using-bdu/-b`), в кеш загружаются **только** базы уязвимостей `trivy` и РЕД ОС. Поэтому при последующих попытках запуска сканирования безопасности с использованием БДУ ФСТЭК России может возникать ошибка вида:

```
   2025-05-13T08:28:44+03:00 ERROR [vulndb] Using flag '--with-bdu'
without BDU cahce file
   2025-05-13T08:28:44+03:00 WARN [vulndb] Please remove vulnerability
database and try again
   2025-05-13T08:28:44+03:00 FATAL Fatal error init error: DB error:
Using flag '--with-bdu' without BDU cahce file
```

Для ее решения необходимо очистить кеш загруженных баз уязвимостей с помощью команды:

```
   trivy clean --vuln-db
```

Затем потребуется снова запустить сканирование с использованием опции `--using-bdu/-b` и дождаться загрузки баз уязвимостей.

4.7 Регистрация событий безопасности

Регистрация событий безопасности производится средствами хостовой ОС.

События в журнале безопасности регистрируются с указанием следующих данных:

- номер (уникальный идентификатор) события, дата, время, тип события безопасности;
- описание события безопасности;
- сведения о критичности события безопасности;
- идентификатор пользователя хостовой ОС, от имени которого был запущен контейнер.

Регистрируемые в журнале события сканирования `docker`-образа отображаются в виде следующих записей:

```
   Jun 23 09:31:30 localhost trivy[46533]: Время=2025-06-23T09:31:30.21-
8490006+03:00 Тип="Сканирование docker-образа" id=d588df6c-bfc4-4c47-
a2f0-f0ba5f7d17c1, Пользователь=0, Уровень=Высокий, Сообщение="В docker-
образе registry.red-soft.ru/ubi7/firebird-3:latest обнаружено уязвимос-
тей 1 (критичных - 0, высокого уровня - 1, прочих - 0); ошибки конфигу-
рации отсутствуют"
   ...
   Jun 23 13:56:30 localhost.localdomain trivy[2673]: Время=2025-06-23T-
13:56:30.461011168+03:00 Тип="Сканирование docker-образа" id=0bad0397-
dfb8-46fa-a2c0-6280418dd509, Пользователь=1000, Уровень=Высокий, Сообще-
ние="В docker-образе registry.red-soft.ru/ubi7/ubi:latest уязвимостей не
обнаружено; обнаружено ошибок конфигурации 2 (критичных - 0, высокого
уровня - 1, прочих - 1)"
   ...
   Jun 23 14:37:51 localhost.localdomain trivy[2892]: Время=2025-06-23T-
14:37:51.554796694+03:00 Тип="Сканирование docker-образа" id=b6906d42-
2a39-412e-b491-563d470f5dd5, Пользователь=0, Уровень=Критический, Сообще-
ние="В docker-образе registry.red-soft.ru/ubi7/pgadmin:4 обнаружено
уязвимостей 4 (критичных - 1, высокого уровня - 2, прочих - 1); обнару-
жено ошибок конфигурации 1 (критичных - 0, высокого уровня - 0, прочих
- 1)"
```

Регистрируемые в журнале события сканирования кластера Kubernetes отображаются в виде следующих записей:

```
   Jun 23 17:05:20 master trivy[49043]: Время=2025-06-23T17:05:20.34819-
2323+03:00 Тип="Сканирование кластера kubernetes"id=416fd4f5-b2aa-4089-
98a3-99f70bf6653e, Пользователь=0, Уровень=Высокий, Сообщение="В docker-
образе registry.k8s.io/kube-proxy:v1.32.4 обнаружено уязвимостей 19
(критичных - 0, высокого уровня - 1, прочих - 18)"
   ...
   Jun 23 17:05:20 master trivy[49087]: Время=2025-06-23T17:05:20.5219-
41561+03:00 Тип="Сканирование кластера kubernetes"id=970c8fec-2589-4649-
b55c-5ce4860ad628, Пользователь=0, Уровень=Высокий, Сообщение="В ресурсе
DaemonSet/kube-proxy обнаружено ошибок конфигурации 23 (критичных - 0,
высокого уровня - 4, прочих - 19)"
   Jun 23 17:05:20 master trivy[49101]: Время=2025-06-23T17:05:20.57466-
```

```
4976+03:00 Тип="Сканирование кластера kubernetes"id=06230558-cde5-4752-
9b0f-5e31606ec64f, Пользователь=0, Уровень=Средний, Сообщение="В ресурсе
Service/kube-dns обнаружено ошибок конфигурации 1 (критичных - 0,
высокого уровня - 0, прочих - 1)"
   ...
   Jun 23 17:05:20 master trivy[49157]: Время=2025-06-23T17:05:20.83755-
7161+03:00 Тип="Сканирование кластера kubernetes"id=47e5728d-ed11-471b-
82e1-7931da6062bc, Пользователь=0, Уровень=Высокий, Сообщение="В ресурсе
Pod/kube-controller-manager-master обнаружено ошибок конфигурации 19
(критичных - 0, высокого уровня - 2, прочих - 17)"
   ...
   Jun 23 17:05:21 master trivy[49199]: Время=2025-06-23T17:05:21.05445-
5718+03:00 Тип="Сканирование кластера kubernetes"id=48b0c9a3-ad1a-4090-
9263-36c8aeb59da4, Пользователь=0, Уровень=Высокий, Сообщение="В ресурсе
Pod/etcd-master обнаружено ошибок конфигурации 16 (критичных - 0,
высокого уровня - 2, прочих - 14)"
```

Журнал событий безопасности сканера безопасности доступен только для чтения. Проверку установленных прав на журнал можно произвести командой:

```
ls -l /var/log/trivy/trivy.log
```

```
-rw-r--r--. 1 root root 2638 июн 23 10:28 /var/log/trivy/trivy.log
```

Стандартная настройка ротации и архивации журнала производится при достижении размера `10 МБ`. Настройка параметров осуществляется только суперпользователем `root` в файле `/etc/logrotate.d/trivy`.

Содержимое файла имеет следующий вид:

```
/var/log/trivy/trivy.log {
   missingok
   maxsize 10M
   compress
   rotate 10
   copytruncate
   delaycompress
   create 0600 root root
}
```

При достижении файлом журнала максимально установленного размера будет выполнено его архивирование. Копия журнала будет сохранена в каталоге `/var/log/trivy` с указанием даты ее создания.

Пример файла:

```
ls -l /var/log/trivy/
```

```
-rw-------. 1 root root  2638 июн 23 10:28 trivy.log
-rw-r--r--. 1 root root 12578 июн 23 10:26 trivy.log-20250623
```