МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«_20_» _02_____ 2026 г.

СИСТЕМНОЕ АДМИНИСТРИРОВАНИЕ И DEVOPS

Методические указания по практическим занятиям и лабораторным работам для студентов направления подготовки «Информатика и вычислительная техника»

Курск 2026

УДК 004.45

Составитель Д.О. Бобынцев

Рецензент: к.т.н., доцент Конаныхина Т.Н.

Системное администрирование и Devops: методические указания к практическим занятиям и лабораторным работам / Юго-Зап. гос. ун-т; сост.: Д.О. Бобынцев. Курск, 2026. – 46 с.

Содержит методические указания по практическим и лабораторным занятиям дисциплины «Системное администрирование и Devops». Даны теоретические материалы, описание порядка выполнения работ, контрольные вопросы, список литературы. Предназначено для студентов направления подготовки «Информатика и вычислительная техника».

Текст печатается в авторской редакции

Подписано в печать 20.02.26. Формат 60x84 1/16.
Усл.печ. л. 2,67. Уч.-изд. л. 2,42. Тираж 100 экз. Заказ. 186 Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

Лабораторные работы

Знакомство с представлением данных, получаемых из различных устройств в Linux

Для выполнения работ вам понадобится виртуальная машина с установленной операционной системой Астра Линукс или Ред ОС. Для создания виртуальной машины рекомендуем использовать бесплатную платформу виртуализации Oracle VM VirtualBox.

Теоретический материал

В этой работе вам предстоит расширить представления о работе с блочными устройствами и файловыми системами, а также познакомиться с менеджером логических томов LVM, который значительно расширяет возможности Linux-администраторов по управлению системами хранения. Рассмотрим архитектуру подсистемы хранения данных в Linux (рис. 1).

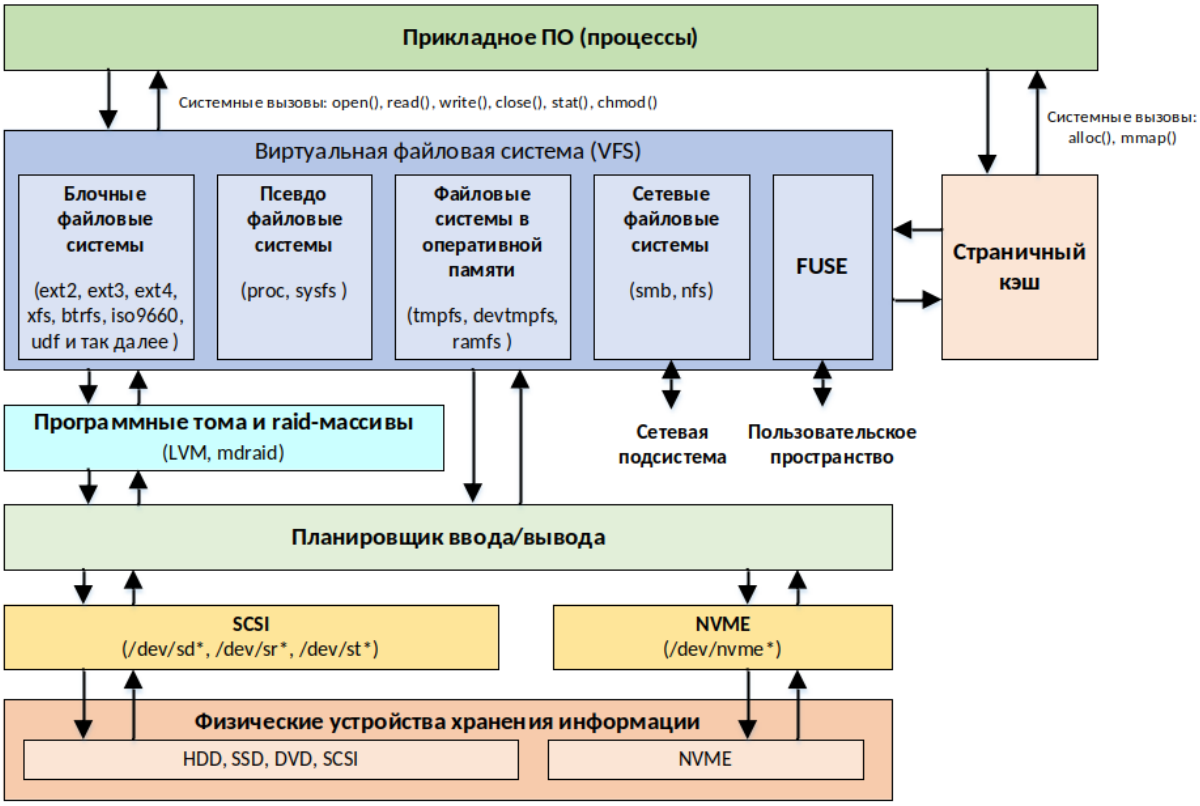

Рисунок 1 – Архитектура подсистемы хранения данных

На верхнем уровне расположены прикладные программы. Они позволяют как управлять подсистемой хранения (создавать, изменять и удалять разделы дисков, форматировать разделы и так далее)

так и работать с данными, хранящимися в файловых системах (создавать файлы, изменять их, менять права, просматривать атрибуты и так далее).

Через системные вызовы прикладное ПО взаимодействует с виртуальной файловой системой (*англ. Virtual File System, VFS*) и со страничным кэшем (*англ. Page Cache*).

Виртуальная файловая система (еще называемая виртуальным коммутатором файловой системы) — это прослойка, уровень абстракции, между системными вызовами и конкретной реализацией файловой системы. Она предназначена для единообразия доступа прикладного ПО к различным типам файловых систем (блочной, в оперативной памяти, сетевой и так далее). VFS декларирует программный интерфейс, позволяющий без внесения изменений в ядро ОС обеспечивать поддержку новых файловых систем.

Страничный кэш – это набор страниц в оперативной памяти, в которые была записана информация с физического устройства хранения для обеспечения быстрого доступа к данным. При каждой операции страничного ввода-вывода, ядро ОС проверяет наличие страницы в кэше, и, по возможности, не использует затратную процедуру обращения к физическому устройству.

Виртуальная файловая система может взаимодействовать с планировщиком ввода/вывода непосредственно или через подсистему программных томов или RAID-массивов (таких как LVM или MDRAID), позволяющих гибко управлять разделами, расширять их, размещать их на нескольких физических устройствах и так далее, без необходимости прерывания работы (без простоя) и без необходимости переконфигурирования физических устройств хранения. Платой за это служит некоторое (небольшое) количество накладных расходов, за еще один уровень абстракции.

Планировщик ввода/вывода – это компонент ядра, который выполняет оптимизацию запросов к диску для повышения производительности и пропускной способности, что особенно критично при работе с жесткими дисками, у которых перемещение считывающей головки является крайне затратной операцией.

Алгоритмов планирования существует несколько, и с помощью команды `sudo dmesg | grep "io scheduler"` вы можете посмотреть, какие из них были зарегистрированы в системе. Команда `cat /sys/block/sda/queue/scheduler` покажет, какой из них используется для конкретного диска.

По умолчанию ALSE использует планировщик mq-deadline, который хорошо подходит для операций асинхронной записи на HDD. Если же у вас SSD, то вам лучше использовать алгоритм kyber, который повышает приоритет операций чтения над записью, или отключить планировщик вовсе.

Физические устройства хранения информации и их файлы

На нижнем уровне находятся физические устройства хранения информации, а в качестве интерфейсов доступа к ним выступают файлы каталога /dev. Файлы блочных устройств, к примеру, именуются следующим образом:

- Файл /dev/sd<N> — дисковые устройства хранения информации с интерфейсами, поддерживающими последовательную передачу данных (SATA, SCSI).
- Файл /dev/hd<N> — дисковые устройства хранения с интерфейсами, поддерживающими параллельную передачу данных (PATA).
- Файл /dev/st<N> — ленточные накопители SCSI.
- Файл /dev/nvme<N> — NVMe накопители.

Для дисковых устройств sd и hd имя задается по маске sd/hd<номер_устройства>[<раздел>], где в качестве номера устройства выступают строчные буквы латинского алфавита (a,b,c,…), а номера раздела может быть как целым положительным числом (порядковый номер раздела на диске), так и отсутствовать (весь диск), например sdb1 – это первый раздел на втором накопителе SCSI.

Диски SSD могут именоваться как sd, если для подключения используют интерфейс SATA, и как nvme, если для подключения используется интерфейс PCI Express. Формат имен nvme довольно сильно отличается от sd/hd устройств, т.к. это не просто быстрые диски, а целая технология, которая была разработана специально для твердотельных накопителей. Имена NVMe устройств задаются в следующем формате:

nvme<номер_контроллера>n<пространство_имён>p<номер_раздела>

Где:

- **nvme<номер_контроллера>** - номер контроллера NVMe, начиная с нуля. Контроллер является символьным устройством.
- **n<пространство_имён>** - пространство имён (*англ. namespace*), нумерация начинается с единицы. Пространства имен

напоминают RAID-массивы, так как могут выходить за пределы одного устройства и создавать логические тома, распределенные по нескольким контроллерам сразу. Это блочное устройство.

- **p<номер_раздела>** – номер раздела в пределах пространства имен, нумерация начинается с единицы. Эти разделы являются полной аналогией с разделами жестких дисков. Это блочное устройство.

Например, nvme0n1p1 – это первый раздел в первом пространстве имен на первом nvme-контроллере компьютера.

В системе Windows во избежание проблем с именами дисков для логических томов, находящихся на дополнительных устройствах, рекомендуется использовать буквы с конца алфавита (X:\, Y:\, Z:\). Это связано это с тем, что автоматическая нумерация устройств зависит не только от того, к каким портам они подключены, но и от внешних факторов, например, как быстро они включаются.

Не следуя этой практике, вы вполне можете столкнуться с тем, что порядок дисков на сервере будет меняться между загрузками, и буквы станут прыгать в хаотичном порядке, как у дислектика, а вместе с ними начнут отваливаться сервисы, использующие эти данные.

В части автоматической нумерации дисков в Linux ситуация ровно такая же – буквы присваиваются дискам автоматически, и вы не можете полагаться на них, т.к. устройства вполне могут устроить день перемены имен.

Файловая система Unix-подобных систем представлена единым пространством имен, что достигается через механизм монтирования различных файловых систем в единую структуру каталогов. Учитывая, что в единой структуре каталогов могут присутствовать совершенно разные файловые системы, потребовался унифицированный интерфейс, который бы обеспечил единообразное взаимодействие со всеми файловыми системами, и таким инструментом стала виртуальная файловая система (*от англ. Virtual File System, VFS*).

Система VFS позволяет с помощью одних и тех же системных вызовов взаимодействовать с любыми типами файловых систем, в том числе и сетевыми, как будто это обычная файловая система ext4 на локальном блочном устройстве. В состав VFS входят следующие типы объектов, которыми она оперирует:

- **суперблок** – метаданные файловой системы;

- **индексные дескрипторы** (*англ. inode*) – метаданные файла;
- **записи каталогов** (*англ. directory entry, dentry*) – специальные файлы, в которых хранятся жесткие ссылки, связывающие имена файлов с их индексными дескрипторами;
- **открытые файлы** – структуры, содержащие информацию об открытых файлах.

Для хранения информации о поддерживаемых типах файловых систем используется таблица, которая создается во время компиляции ядра. Каждая запись этой таблицы включает наименование типа и указатель на функцию, вызываемую во время монтирования этой файловой системы (если значение не указано, используется функция монтирования по умолчанию).

Функция монтирования используется для считывания суперблока, установки внутренних переменных и возврата дескриптора смонтированной системы обратно в VFS. После того, как система смонтирована, функции VFS используют открытый дескриптор для доступа к процедурам чтения/записи этой файловой системы.

Для более эффективного управления памятью и устранения значительной фрагментации данных система VFS кэширует метаданные, используемые при монтировании (суперблок, индексные дескрипторы, записи каталогов и др.), с помощью специального механизма управления памятью, который называется «распределение slab» (*англ. slab allocation*). Секрет быстродействия этого алгоритма состоит в повторном использовании памяти, освобожденной одним объектом, для размещения другого объекта того же типа.

Управление кэшированием метаданных осуществляется с помощью переменной ядра ОС `vm.vfs_cache_pressure` в файле `/proc/sys/vm/vfs_cache_pressure`, значение которой представляет из себя целое число:

- `0` – ядро сохраняет (не освобождает) страничный кэш, может привести к нехватке оперативной памяти.
- `100` – ядро пытается сохранить справедливое распределение памяти между кэшем страниц (pagecache) и кэшем подкачки (swapcache). Используется по умолчанию.
- `>100` – ядро активно выгружает метаданные в VFS. При значении существенно выше 100 это может негативно сказаться на производительности VFS.

Текущую статистику по использованию кэшей slab можно посмотреть утилитой sudo slabtop (рис. 2).

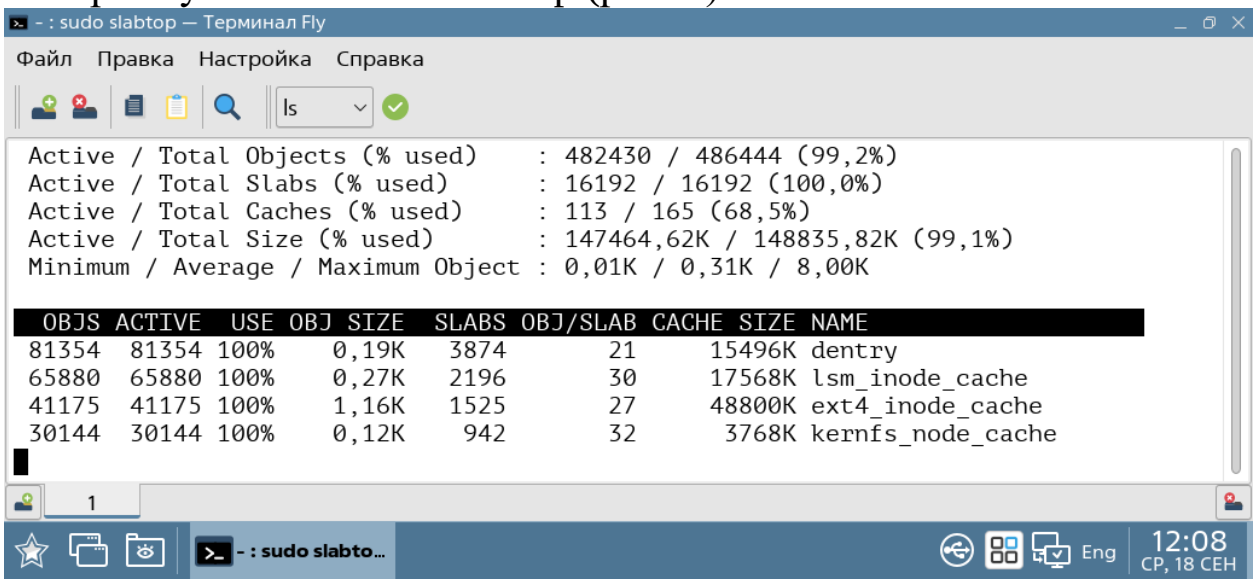

Рисунок 2 – Утилита slabtop

Для принудительного сохранения «грязных» страниц кэша (*англ. dirty pages*), которые были изменены после их помещения в кэш, можно воспользоваться утилитой sync. Она гарантирует, что все изменения в памяти будут записаны на диск, предотвращая потерю изменений в кэше при аварийном завершении работы системы.

Если же вам потребуется освободить память, зарезервированную под кэш, вы можете из-под суперпользователя выполнить команду echo 3 | sudo tee /proc/sys/vm/drop_caches

Операционная система Linux способна работать с разными типами файловых систем, как поддерживающими концепцию индексных дескрипторов (ext4, xfs, BtrFS), так и совершенно чуждые им (exFAT, NTFS). Список поддерживаемых файловых систем определяется драйверами, то есть модулями, которые были включены в состав ядра и загружены в данный момент.

Список всех модулей ядра с драйверами файловых систем можно получить с помощью команды ls /lib/modules/$(uname -r)/kernel/fs

Список файловых систем, поддерживаемых ядром в данный момент, можно получить командой cat /proc/filesystems. Словом, «nodev» (*англ. no device*) обозначаются файловые системы, которые не привязаны к устройствам, но это не означает, что данная система не используется, т.к. в эту категорию попадают псевдофайловые системы и системы, размещаемые в памяти.

Для Linux родными считаются файловые системы из семейства Extended File System, EXT. Их разработка началась в начале 90-х для преодоления ограничений предшествующей им Minix File System как по длине имен файлов, так и по размеру поддерживаемых дисков.

К настоящему времени разработано несколько редакций, каждая из которых становилась значительно лучше предыдущей. В 2008 году была представлена четвертая и крайняя на текущий момент версия ext4, отличительными особенностями которой являются:

- Максимальный размер файла до 16 Тебибайт, 2^{40} байт (против 2 ТБ для ext3).
- Максимальный размер тома: до 1 Эксбибайта, 2^{60} байт (против 32 ТБ для ext3).
- Максимальное число подкаталогов в каждом каталоге 64 000 (против 32 000 для ext3).
- Максимальная длина имени файла 255 байт (или 127 русских символов в utf8).
- Журналируемая ФС, что помогает сохранить целостность файловой системы при сбоях.
- Использует контрольные суммы для автоматического обнаружения повреждения данных из-за аппаратных сбоев. КС добавлены для суперблока, inode, битовых карт блоков, блоков дерева экстентов, htree-узлов, журналов и др.
- Обратно-совместима к ext2 и ext3, что позволяет монтировать старые файловые системы через драйвер ext4.
- Поддерживает **пространственную запись файлов** – резервирование на диске всего требуемого пространства перед началом записи файла на диск.
- Поддерживает **экстенты** – хранение в дескрипторе адресов только первого и последнего блока из последовательного массива блоков данных. Размер экстента может достигать 128 МБ, что позволяет существенно повысить производительность ФС.
- Уменьшает фрагментацию данных за счет более рационального выделения блоков.
- Поддерживает отложенное выделение блоков памяти (непосредственно перед их записью на диск), что позволяет снизить нагрузку на кэш и увеличить производительность.

- Для работы с каталогами использует разновидность B-дерева, что позволяет без снижения производительности работать с большим количеством каталогов.
- Поддерживает быструю проверку занятых блоков, исключая из проверки индексные дескрипторы и свободные блоки данных.

К недостаткам ext4 можно отнести только отсутствие поддержки таких функций как дедупликация данных и снижение производительности при работе с разделами более 100 ТБ.

Хотя файловые системы семейства ext и отличаются друг от друга в деталях, но их структура практически идентична (рис. 3).

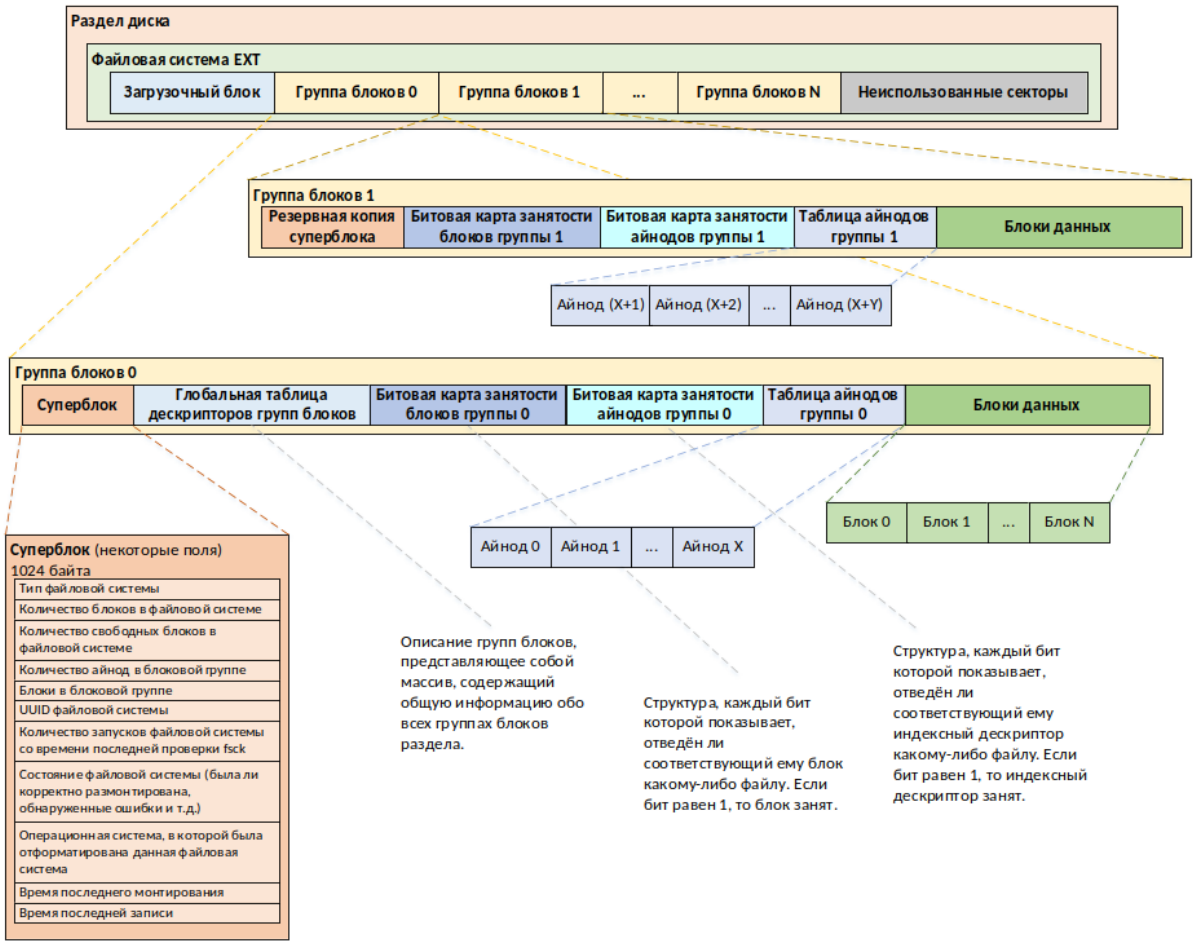

Рисунок 3 – Структура системы ext

В системе Linux процедура разметки раздела для работы с конкретной файловой системой называется созданием файловой системы, но более привычный для Windows-администраторов термин «форматирование диска» тоже используется.

Как мы помним, в начале раздела часть секторов выделяется под размещение загрузочного блока, а после него размещаются дан-

ные. Для уменьшения фрагментации и повышения производительности файловая система ext4 нарезает все доступное пространство на фрагменты, которые называются группами блоков.

В файловой системе NTFS нет аналога для групп блоков, так как каждая группа в какой-то степени является еще одной файловой системой, у которой есть собственные блоки данных, таблица файловых дескрипторов и дополнительная метаинформация. Маленькие файлы, конечно, не должны выходить за пределы одной группы, но если нам потребуется сохранить большой архив, то он без проблем оккупирует несколько групп сразу.

Первая группа блоков, которая идет под номером ноль, немного отличается от всех остальных и содержит дополнительные метаданные. В нулевую группу блоков входят следующие разделы:

- **Суперблок** – это структура данных размером 1024 байта, в которой содержится информация о файловой системе. Для обеспечения надежности файловая система создаёт еще несколько резервных копий суперблока, и хранит их внутри других групп блоков в разных частях диска. Суперблок содержит следующую информацию:
  - Тип файловой системы (s_type).
  - Размер файловой системы в логических блоках, включая сам суперблок, ilist и блоки хранения данных (s_fsize).
  - Размер массива индексных дескрипторов (s_isize).
  - Число свободных блоков, доступных для размещения (s_tfree).
  - Число свободных индексных дескрипторов, доступных для размещения (s_t inode).
  - Состояние файловой системы (флаг модификации s_fmod, флаг режима монтирования s_fronly).
  - Размер логического блока в виде числа, обозначающего степень 2 (9 – 512 байт, 10 – 1КБ).
  - Количество логических блоков в группе блоков.
  - Количество индексных дескрипторов в группе блоков.
  - Идентификатор файловой системы (UUID).
  - Время последней записи.
  - Время последнего монтирования.
- **Глобальная таблица дескрипторов групп блоков** — это массив, в котором содержится информация о расположении всех групп блоков раздела.

- **Битовые карты занятости блоков и индексных дескрипторов** — помогают быстро определять, какие из блоков и индексных дескрипторов заняты, а какие свободны. Они представляют из себя массив двоичных данных, где каждый бит соответствует блоку данных или индексному дескриптору, а значение показывает занят он (1) или свободен (0).
- **Таблица индексных дескрипторов** — содержит массив индексных дескрипторов. Структура индексного дескриптора подробно рассматривалась в модуле про работу с файлами. Стоит отметить, что индексные дескрипторы создаются для всего раздела в целом и их нумерация уникальна для всех групп блоков.
- **Блоки данных** — объединяют группу секторов диска для хранения содержимого файлов, каталогов, косвенной адресации. Размер блоков данных кратен размеру сектора, который в большинстве случаев составляет 512 байт, но есть диски с секторами и по 4 Кб. Блоки данных по своей сути ближе всего к кластерам NTFS.

Остальные группы блоков содержат следующие разделы:

- Битовая карта занятости блоков группы 0
- Битовая карта занятости индексных дескрипторов группы 0
- Таблица индексных дескрипторов группы 0
- Блоки данных

Некоторые группы блоков могут содержать резервную копию суперблока.

В Linux у дисков есть разделы, но мы с ними не работаем как с логическими дисками Windows. Файловая система раздела может быть смонтирована в любую точку единой файловой системы, которая соответствует системному диску, с которого была выполнена загрузка. Точно также монтируются внешние носители информации, такие как USB-флешки, DVD-диски.

Вы можете использовать для работы сервера один раздел диска, но в высоконагруженных системах некоторые каталоги рекомендуется выносить отдельно по следующим причинам:

Каталог	**Для чего используется**	**Почему может быть выгодно перенести в другой раздел**	**Рекомендуемый размер**
/home	Пользовательские файлы	Домашние каталоги пользователей, аналог C:\Users в Windows. Перенос на другой раздел диска помогает исключить переполнение	Не менее 100 МБ

Каталог	Для чего используется	Почему может быть выгодно перенести в другой раздел	Рекомендуемый размер
		системного диска и гарантирует сохранность файлов при переустановке.	
/tmp	Временные файлы	Каталог хранит временные файлы. Перенос на другой раздел диска помогает исключить переполнение системного диска.	Не менее 50 МБ
/usr,/opt	Файлы программ	В этих каталогах хранятся файлы программ, аналог Program Files Если система работает на быстром SSD, то программы можно вынести на HDD большого объема.	Не менее 8 ГБ
/var	Динамические данные	Этот каталог используется для динамических данных, таких как файлы журнала, кэшированные данные и т.д. Перенос на другой раздел диска помогает исключить переполнение системного диска	Не менее 400 МБ
swap	Раздел подкачки	Раздел подкачки необходим для увеличения объема доступной памяти, аналог файла подкачки в Windows. Перенос на другой раздел диска позволяет повысить производительность, если это SSD диск, и уменьшить фрагментацию файла подкачки.	Выбирается исходя из объема оперативной памяти. Если необходима гибернация, то не менее объема оперативной памяти + небольшой запас в 1-2 ГБ.

Однако, следует учитывать, что при разбиении диска на разделы мы можем ошибиться в своих предположениях о том, сколько места потребуется на те или иные задачи, и поменять это решение «на лету» уже не получится.

Решением этой проблемы является использование дополнительного уровня абстракции в виде системы логических томов, работающих поверх дисковых разделов. Универсальным решением являются менеджер логических томов LVM.

Перед практическим изучением работы утилит, необходимо добавить несколько виртуальных дисков в виртуальную машину: 3 диска по 1 ГБ и один диск на 2 ГБ.

Для управления дисковыми разделами, в Astra Linux доступно несколько утилит:

1. Утилита fdisk – консольная утилита для разметки диска (создания и удаления разделов).

2. Утилита parted – консольная утилита для разметки диска (создания, удаления и изменения разделов).

3. Утилита sfdisk – утилита для использования в скриптах;

4. Утилита cfdisk – псевдографическая утилита работы с разделами.

5. Утилита gparted – графическая утилита разметки диска.

После добавления дисков посмотрим список устройств хранения и их разделы с помощью команды sudo fdisk –l (L строчная). Для вывода информации только об одном диске необходимо к команде добавить имя диска, например, sudo fdisk -l /dev/sdb. Утилита fdisk может работать также в интерактивном режиме. Для этого необходимо просто указать имя диска sudo fdisk /dev/sdb. Результат будет таким:

Добро пожаловать в fdisk (util-linux 2.33.1).

Изменения останутся только в памяти до тех пор, пока вы не решите записать их.

Будьте внимательны, используя команду write.

Устройство не содержит стандартной таблицы разделов.

Создана новая метка DOS с идентификатором 0x68735955.

Команда (m для справки):

Если вы ответите m и нажмёте Enter, терминал выведет полную справку по дальнейшей работе с командой. Создадим новый раздел, используя команду n (при этом на первые 3 вопроса ответы можно опустить – нас устроит значение по умолчанию):

n (создать раздел) -> [p] (первичный) -> [1] (номер раздела) -> [2048] (первый сектор раздела, традиционно место в начале диска зарезервировано под код первичного загрузчика и «пустые» сектора).

Затем утилита попросит ввести последний сектор раздела (значение по умолчанию – самый последний сектор, весь диск). Кроме номера сектора, можно использовать выражения +/-<сектора> или +/-<размер>{K,M,G,T,P}, где K, M, G, T и P – это килобайты, мегабайты, и так далее. А мы создадим раздел размером в 200 МБ. Для этого введем +200M:

Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-2097151, default 2097151): +200M

Создан новый раздел 1 с типом 'Linux' и размером 200 MiB

Команда (m для справки):

Введем команду p и выведем информацию о разделах на диске.

С помощью остальных команд можно менять типы разделов, и удалять разделы. Работа с командой fdisk может быть завершена без сохранения изменений командой q и с записью на диск командой w.

Для управления файловыми системами есть несколько команд:

mkfs.<тип_файловой_системы> [<параметры>] <устройство> – создание файловой системы

mkswap <устройство> – создание раздела подкачки

resize2fs <устройство> <размер> – изменение размера раздела файловой системы

Команда mkfs использует настройки по умолчанию, которые заданы в файле /etc/mke2fs.conf. Обратите внимание, что некоторые команды mkfs являются ссылками на другие утилиты.

С параметрами утилиты можно ознакомиться в справке, но стоит выделить параметр -i, который задает количество байт информации на один индексный дескриптор, что в свою очередь определяет, сколько места на диске будет отдано под индексные дескрипторы. Так как для файловых систем ext количество индексных дескрипторов жестко задается при создании ФС, стоит сразу на этапе её создания определиться с их числом.

Создадим файловую систему ext4 на разделе диска /dev/sdb с параметрами по умолчанию командой sudo mkfs.ext4 /dev/sdb2. Теперь немного расширим этот раздел (~100МБ) и расширим файловую систему:

sudo parted /dev/sdb resizepart 2 400MiB

lsblk

sudo resize2fs /dev/sdb2

Для проверки целостности файловой системы служит команда fsck (англ. file system check). Для её успешной работы раздел не должен быть смонтирован. Запустим её для раздела /dev/sdb2: sudo fsck /dev/sdb2

С помощью графической утилиты gparted, имеющей интуитивно понятный интерфейс, можно как разметить диск, так и создать на его разделах файловые системы, а при необходимости изменить их размеры и атрибуты.

Утилита Gparted может быть запущена из меню Пуск ‣ Системные ‣ Редактор разделов Gparted или из командной строки (sudo gparted).

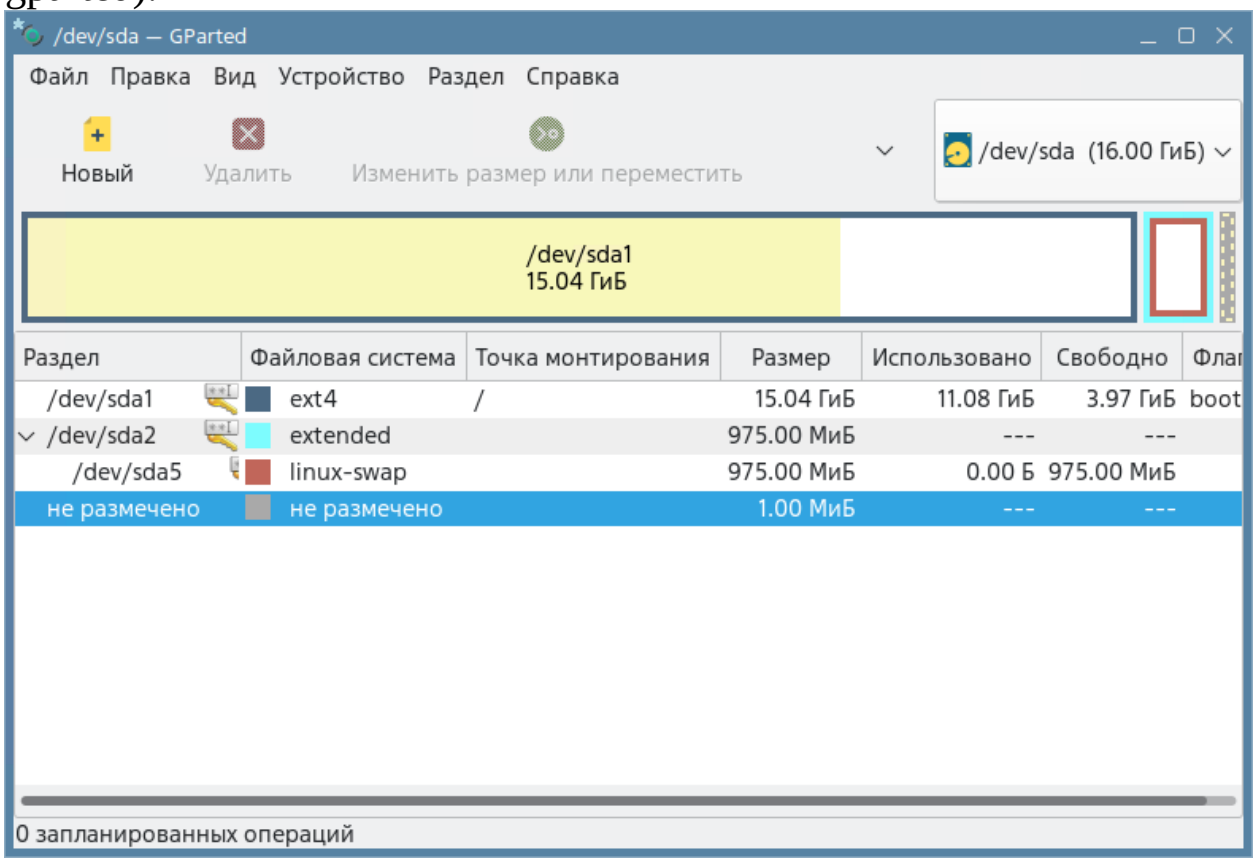

Рисунок 4 – Утилита Gparted

Выберем в правом верхнем углу диск /dev/sdb, в нем раздел sdb2, и применим действие Изменить размер или переместить. Переместим этот раздел в начало диска. Согласимся с предупреждением утилиты. Теперь создадим новый раздел в свободной области диска размером в 500МБ. Раздел был создан и осталось свободное место.

Заметьте, что фактических операций с диском еще не проводилось, они были только запланированы. Для применения изменений, необходимо выбрать пункт Применить все операции в меню Правка. Применим операции, подтвердив их. В процессе применения изменений вы будете видеть статус операции, а после их завершения отчет о выполнении.

Мы познакомились с тем, как в Linux можно разметить диски и создать на них файловые системы. Однако, для доступа к файловым системам этого недостаточно. Как в Windows мы не можем стандартными средствами обратиться к данным на диске, если ему не назначена буква, так и в Linux мы не можем обратиться к файловой системе, если она не была смонтирована.

Чтобы данные раздела диска стали доступны, этот раздел должен быть смонтирован в корневую файловую систему. Процесс монтирования файловой системы с точки зрения оператора ПК не очень сложен.

Существуют следующие виды монтирования:

- Временное через консольную команду mount.
- Постоянное через файл /etc/fstab.
- Монтирование с помощью службы systemd.

Опции монтирования:

- **auto** – файловая система монтируется автоматически.
- **ro** – монтируется в режиме «только чтение».
- **rw** – монтируется в режиме «чтение и запись».
- **dev** – файловая система может содержать файлы блочных и символьных устройств.
- **exec** – файловая система может содержать исполняемые файлы.
- **suid** – разрешено использование битов SUID и SGID.
- **user** – разрешено обычному пользователю размонтировать данную файловую систему и пр этом используются значения по умолчанию (noexec, nosuid, nodev).
- **defaults** – установки по умолчанию (rw, suid, dev, exec, nouser, async).
- **codepage=кодовая_страница** – применять указанную кодировку к именам файлов.
- **iocharset=набор_символов** – отображать имена файлов в соответствии с указанным набором символов.
- **noauto** – файловая система не может быть автоматически смонтирована.
- **nodev** – файловая система не может содержать файлы блочных и символьных устройств.
- **noexec** – файловая система не позволяет запускать исполняемые файлы.
- **nosuid** – запрещено использование битов SUID и SGID.
- **nouser** – обычному пользователю запрещено размонтировать данную файловую систему.

Временное монтирование выполняется утилитой mount и действует до очередной перезагрузки системы. Программа mount способна самостоятельно определить тип файловой системы, либо тип

можно указать явно с помощью опции -t. Для определения типа файловой системы утилита mount использует библиотеку *blkid*.

Для вывода информации обо всех примонтированных в данный момент ФС и их параметров монтирования можно вызвать утилиту mount с ключом -l (строчная L) или без параметров вовсе.

Для отображения информации только о реальных файловых системах можно воспользоваться фильтрацией вывода через утилиту grep mount | grep "^/dev".

Более наглядный вывод дает утилита findmnt. Чтобы с ее помощью получить список реальных файловых систем, можно использовать ее с ключом findmnt --real.

Вызов команды findmnt с опцией -p позволит следить за процессом монтирования и размонтирования в реальном времени. Откроем новую вкладку в терминале и запустим там эту команду findmnt -p.

Создадим в каталоге /mnt подкаталог data и примонтируем в него устройство /dev/sdb2 с опциями по умолчанию. Синтаксис команды mount при монтировании:

sudo mount [<опции>] <устройство> <точка_монтирования>.

Теперь перемонтируем файловую систему с опцией только на чтение. Для этого выполним команду:

sudo mount -o remount,ro /dev/sdb2 /mnt/data

Попробуем прочитать файл /mnt/data/test.txt и внести в него изменения: sudo vim /mnt/data/test.txt. Как мы можем убедиться, теперь даже из-под суперпользователя мы не можем вносить изменения в эту файловую систему. Однако, журнал для ФС ext3 и ext4 может все еще работать. Для отключения возможности записи и в журнал, необходимо добавить при монтировании еще одну опцию noload.

Для размонтирования устройства служит команда sudo umount <точка_монтирования> или sudo umount <устройство>. Для постоянного монтирования (или долговременного) необходимо вносить изменения в конфигурационный файл /etc/fstab.

Формат файла представляет из себя таблицу с полями, разделенными пробелами. Строки, начинающиеся с символа решетки, являются комментариями, а пустые строки – игнорируются.

Всего используется шесть полей:

- **File system (fs_spec) -**– имя, метка (LABEL) или идентификатор (UUID) устройства. Указание метки или идентификатора

предпочтительнее, так как порядок обнаружения оборудования может быть изменен, что приведет к изменению имени устройства.

- **Mount point (fs_file)** –- точка монтирования (если нет, то значение none, как например у раздела подкачки). Если точка монтирования содержит пробелы или табуляции, их можно использовать как «\040» и «\011» соответственно.
- **Type (fs_vfstype)** – предполагаемый тип файловой системы. Можно указать несколько значений через запятую.
- **Options (fs_mntops)** -–- параметры монтирования.
- **Dump (fs_freq)** –- используется утилитой dump. Определяет, какие файлы системы нужно выгружать при дампе. По умолчанию 0 (не выгружать).
- **Pass (fs_passno)** –- используется утилитой fsck. Определяет порядок проверки файловых систем при вызове утилиты fsck. Для корневой ФС этот параметр должен быть 1. Для остальных ФС равен 2 или 0 (не проверять). Системы на разных дисках будут проверяться параллельно, а на одном диске – последовательно.

Менеджер логических томов

Менеджер логических томов LVM (англ. Logical Volume Manager) – это подсистема управления логическими томами, позволяющая использовать разные области одного жесткого диска и/или области с разных жестких дисков как один логический том.

С большой натяжкой аналогом LVM в Windows можно назвать динамические диски, которые тоже позволяют создавать программные RAID массивы и объединять несколько разделов на разных физических носителях в один логический раздел.

LVM основан на Device mapper, которая используется также в реализации программных RAID-массивов, системы шифрования дисков dm-crypt и создания снимков файловой системы.

Основные преимущества LVM:

- Одну группу логических томов можно создавать поверх любого количества физических разделов.
- Размер логических томов можно легко менять прямо во время работы.
- LVM поддерживает механизм снимков состояний (*англ. snapshot*), копирование разделов «на лету» и зеркалирование, подобное RAID-1 и чередующуюся запись, подобно RAID-0.

К сожалению, получая преимущества приходиться чем-то жертвовать, и LVM не исключение. Удобство и гибкость при работе с LVM мы разменяваем на следующие недостатки:

- Работа дополнительно «прослойки» и уровня абстракции требует дополнительных накладных расходов, что может снизить производительность дисковой подсистемы (в среднем потери можно оценить в 5-10%)
- При использовании LVM на нескольких дисках, при потере одного из них, все данные хранимые на LVM будут утеряны.
- Начальная настройка LVM более сложна, чем просто разбиение диска. Необходимо понимать терминологию и модель LVM (логические тома, физические тома, группы томов), прежде чем вы сможете начать его использовать.

В состав менеджера входят следующие компоненты (рис. 5):

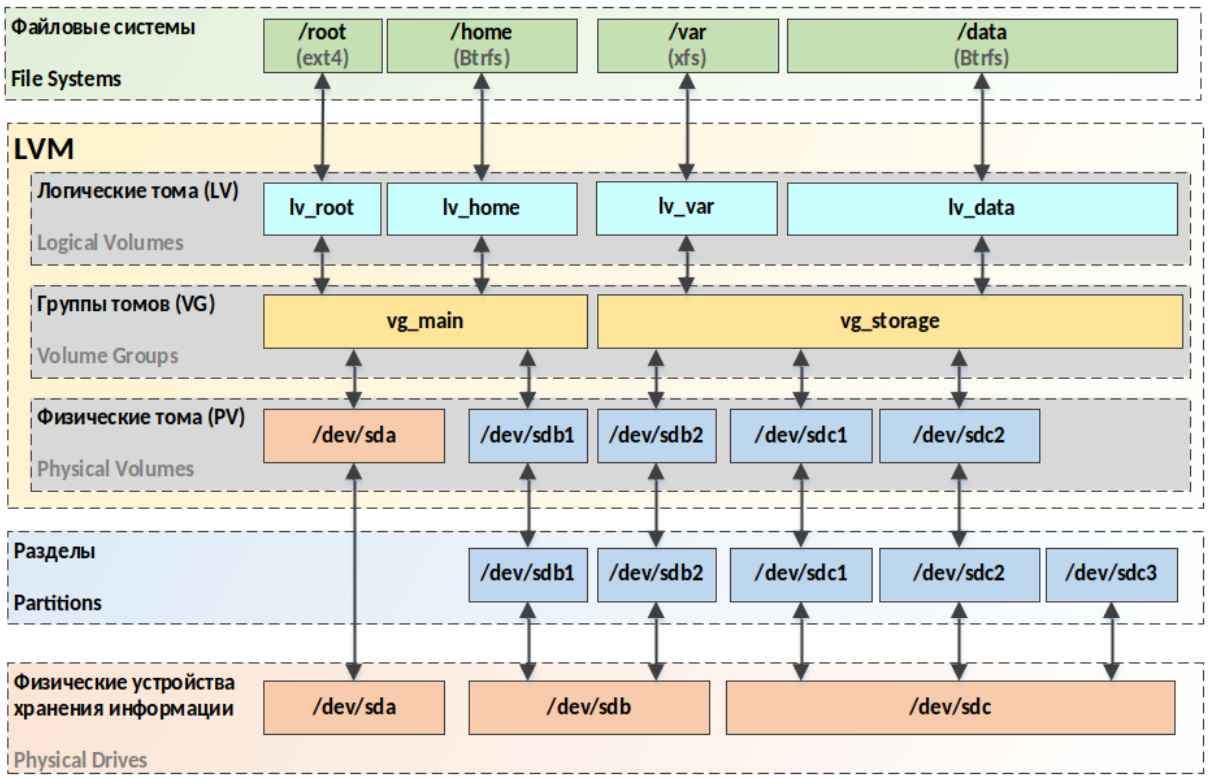

Рисунок 5 – Менеджер логических томов

- **Физический том** (*англ. Physical Volume*, PV) –- раздел на диске или весь диск. В том числе, устройства программного и аппаратного RAID (которые уже могут включать в себя несколько физических дисков). Физические тома входят в группу томов.
- **Группа томов** (*англ. Volume Group, VG*) –- это самый верхний уровень абстрактной модели, используемой системой LVM.

С одной стороны, группа томов состоит из физических томов, с другой – из логических и представляет собой единую административную единицу.

- **Логический том** (*англ. Logical Volume, LV*) –- раздел группы томов, эквивалентен разделу диска в не-LVM системе. Представляет собой блочное устройство и, как следствие, может содержать файловую систему.

Кроме указанных понятий существуют также логический (LE, Logical extent) и физический (PE, Physical extent) экстенты (порции данных). Физические тома делятся на порции данных, равные размеру физическим экстентам, а логические тома на порции данных равные размеру логическим экстентам, размер логического экстента не меняется в пределах группы томов и равен нескольким мегабайтам. При этом существует два алгоритма отображения физических экстентов на логическом:

- **Линейное отображение**, когда физические экстенты последовательно назначаются логическим экстентам.
- **Расслоенное отображение** (*англ. stripped*), когда порции данных логических экстентов разделяют на определенное количество физических томов (похожая схема используется в RAID 0). Это может увеличить производительность работы логического тома, однако логический том с таким отображением не может быть расширен за пределы физических томов, на которых он изначально был создан.

Для работы с LVM необходимо установить пакет lvm2, который по умолчанию не установлен: sudo apt install lvm2 -y. Как вы можете заметить, при этом создаётся новые сервисы и образ Initrd включающий необходимые модули для работы с LVM. В своих экспериментах мы воспользуемся ранее добавленными дисками /dev/sdc, /dev/sdd и dev/sde.

Полный процесс создания LVM и подготовки её к использованию можно представить шагами:

- Создание физических томов
- Создание групп томов
- Создание логических томов
- Создание ФС на логических томах
- Монтирование ФС

Для создания физического тома PV (инициализации целого диска или его раздела) используется команда:

sudo pvcreate <устройство1> [<устройство2> <…> <устройствоN>]

где устройство1 .. устройство<N> — это имя устройства целиком (например, /dev/sdb) или его раздела (например, /dev/sdb1).

Для вывода информации о физических томах существуют следующие команды:

1. Команда pvs –- позволяет настроить формат вывода, показывая по одному тому в каждой строке.
2. Команда pvdisplay –- формирует подробный отчет для каждого физического тома, включая информацию о размере, экстентах, группе томов и пр. Формат вывода фиксирован.
3. Команда pvscan –- проверяет все поддерживаемые блочные устройства в системе на предмет наличия физических томов.

Для удаления физических томов используется команда pvremove <имя_PV>.

После создания физических томов мы можем создать группу томов. Соглашение об именовании предписывает начинать имя группы с префикса «vg_» и использовать либо порядковый номер, либо (что лучше) значащее имя, отражающее назначение этой группы.

Для создания группы томов служит команда sudo vgcreate <имя_группы> <имя_физического_тома1> [<имя_физического_тома2> <имя_физического_...> <имя_физического_N>]

Для вывода информации о группах томов служат команды, аналогичные тем, что используются для вывода информации о физических томах:

1. Команда vgs – позволяет настроить формат вывода, показывая по одной группе в каждой строке.
2. Команда vgdisplay – формирует подробный отчет для каждой группы томов, включая информацию о размере, количестве физических и логических томов и пр. Формат вывода фиксирован.
3. Команда vgscan – проверяет все поддерживаемые дисковые устройства в системе на предмет наличия физических томов и групп томов.

Для добавления нового тома в группу томов используется команда:

sudo vgextend <имя_группы> <имя_тома>

Для удаления тома из группы исспользуется командой:

sudo vgreduce <имя_группы> <имя_тома>

Для переименования группы используется команда:

vgrename <текущее_имя> <новое_имя>

Для удаления группы томов служит команда sudo vgremove <имя_группы>.

Для создания логического тома используется команда:

sudo lvcreate [<опции>] <размер_тома> -n <имя_тома> <имя_группы>

Где размер тома может быть задан несколькими способами:

-L<число> - размер в мегабайтах;

-L<число><единицы_объема> - размер в указанных единицах объема;

-l <число> - размер в логических экстентах;

-l100%FREE – весь объем группы.

Команда sudo lvcreate -i2 -I4 -l100 -n <имя_тома> <имя_группы> создаст логический том размером в 100 логических экстентов с расслоением по двум физическим томам и размером блока 4 KB.

Задание

1. Добавьте к существующей виртуальной машине новый жесткий диск размером 500 Мб.

2. Установите метку диска msdos утилитой parted в режиме командной строки.

3. Создайте на подключенном жестком диске 2 раздела по 100 Мб утилитой parted в режиме командной строки.

4. Создайте на новых разделах файловые системы EXT3 и XFS и смонтируйте их в каталоги /mnt/<part1> и /mnt/<part2>, где part1 - ваша фамилия, part2 - ваше имя.

5. Создайте постоянное монтирование для указанных разделов со следующими условиями:

/mnt/part1 монтируется в режиме только для чтения;

/mnt/part2 монтируется с запретом на запуск исполняемых файлов.

6. Перезагрузите ОС для проверки созданных настроек.

7. После проверки настроек удалите созданные точки монтирования и разделы.

Контрольные вопросы

1. Перечислить составляющие архитектуры подсистемы хранения данных.
2. Что такое виртуальная файловая система?
3. Что такое страничный кэш?
4. Какие типы объектов входят в состав VFS?
5. Какие вы знаете опции монтирования?
6. Что такое менеджер томов LVM?
7. Какие компоненты входят в LVM?

Использование оркестратора Ansible

Для выполнения работы понадобятся три виртуальные машины с установленной операционной системой Ред ОС.

Теоретический материал

Ansible – это система управления конфигурациями, предназначенная для автоматизации настройки и развертывания программного обеспечения. Она поддерживает управление сетевыми устройствами и серверами, на которых установлен Python версии 2.4 и выше, используя SSH или WinRM для установления соединения.

Основное преимущество **Ansible** заключается в его простоте и эффективности. Администратору достаточно описать желаемую конфигурацию системы с помощью специальных сценариев, называемых плейбуками (**playbooks**). Плейбуки пишутся на декларативном языке разметки **YAML**. Благодаря этому, процесс переконфигурирования системы становится быстрым и удобным: для внесения изменений достаточно добавить или изменить несколько строк в сценарии.

Ansible также отличается отсутствием необходимости установки дополнительного программного обеспечения на управляемые узлы, что упрощает его внедрение и использование в различных инфраструктурах.

Для установки **Ansible** необходимо выполнить в терминале следующую команду (потребуются привилегии **администратора**):

```
sudo dnf install ansible
```

Управляемые узлы в Ansible определяются в специальном файле инвентаризации, который по умолчанию расположен по пути /etc/ansible/hosts. Открыть файл на редактирование можно с помощью команды (потребуются права администратора):

```
nano /etc/ansible/hosts
```

В данном файле адреса узлов могут быть указаны в различных форматах: как в виде IP-адресов, так и в виде доменных имён или имён хостов. Например:

```
[test]
node.example.ru
192.168.0.100
```

Указывать адреса узлов можно с помощью функции диапазона, чтобы не вписывать их по одному. Например, чтобы указать десять

узлов (node01, node02, … node10), допускается использование следующей записи:

node[01:10].example.ru

Ещё одна полезная функция – возможность определять псевдонимы для узлов. Например, можно использовать псевдоним custom_name, если указать его в файле следующим образом:

custom_name ansible_host=node.example.ru

Узлы можно объединять в группы. Для этого в квадратных скобках указывается название группы, а ниже перечисляются узлы, входящие в группу:

[group1]
node1.example.ru
192.168.0.100
[group2]
node2.example.ru
192.168.0.101

Для подключения к управляемым узлам используется протокол SSH с использованием RSA-ключей. Необходимо сгенерировать и распространить ключ на все управляемые узлы. Для генерации ключа используется следующая команда:

ssh-keygen -C "$(whoami)@$(hostname)-$(date -I)"

Далее нужно распространить ключ на все подключенные хосты. Распространить ключи на хосты можно командой:

ssh-copy-id user@host

где:

user – пользователь управляемого узла;

host – адрес управляемого узла.

Пример:

ssh-copy-id root@192.168.0.100

Для проверки подключения всех узлов можно использовать встроенный модуль Ansible – ping:

ansible all -m ping

Если для подключения используется SSH с аутентификацией по паролю, то на управляющей машине дополнительно должен быть установлен пакет sshpass. Установить его можно с помощью команды:

sudo dnf install sshpass

Также для аутентификации по паролю необходимо использовать ключ -k при выполнении команды:

ansible all -m ping -k

Об успешном подключении к узлу проинформирует зелёная надпись следующего вида:

```
192.186.0.100 | SUCCESS => {
   "ansible_facts": {
      "discovered_interpreter_python": "/usr/bin/python3.8"
   },
   "changed": false,
   "ping": "pong"
}
```

Для проверки подключения отдельной группы можно указать её название в команде вместо слова «all»:

ansible group_name -m ping

где group_name – это группа хостов, указанная в файле hosts. В результате под каждым хостом должно быть написано "ping": "pong".

При выполнении некоторых команд Ansible для перечисления управляемых узлов можно использовать собственные файлы инвентаризации. Для использования собственных файлов инвентаризации в командах необходимо указывать ключ -i:

ansible all -i /путь/к_файлу/инвентаризации -m ping

Допускается использование нескольких файлов инвентаризации в одной команде с указанием аргумента -i для каждого:

ansible all -i inventory1 -i invetory2 -i inventory3 -m ping

Также возможно перечисление нескольких узлов в одной команде. Нужные узлы перечисляются через запятую:

ansible all -i host1,host2,host3 -m ping

Ещё одной важной функциональной возможностью является поддержка выполнения команды для целого каталога, содержащего файлы инвентаризации. Для реализации данного подхода необходимо создать соответствующую иерархическую структуру файлов, которая должна быть организована следующим образом:

```
inventory/
        inventory1
        inventory2
        inventory3
```

В данном случае при выполнении команды достаточно указание каталога, содержащего файлы инвентаризации. В результате команда будет автоматически выполнена для всех узлов, перечисленных в файлах указанного каталога:

ansible all -i inventory -m ping

Ansible позволяет выполнять одиночные bash-команды без создания плейбуков. Например, для проверки объема и использования оперативной памяти на управляемых узлах выполните команду:

ansible all -a "free -h"

В результате выполнения команды управляемые узлы могут вернуть сообщение следующего вида:

```
192.168.0.100 | CHANGED | rc=0 >>
               total        used        free      shared  buff/cache   available
Mem:           954Mi       282Mi       147Mi       6,0Mi       524Mi       519Mi
Swap:          2,0Gi       162Mi       1,8Gi
```

Плейбуки (playbooks) — это сценарии, написанные на декларативном языке разметки YAML, которые выполняются на управляемых узлах. Благодаря своей декларативной природе плейбуки позволяют задавать желаемую конфигурацию узла в виде набора задач.

Файлы плейбуков можно хранить в любом удобном месте, но важно, чтобы файлы имели расширение .yml. Их синтаксис зависит от отступов строки! Для создания плейбука можно выполнить следующую команду:

nano playbook.yml

В качестве первого примера рассмотрим создание плейбука для вывода переменной $HOSTNAME со всех управляемых узлов с помощью выполнения bash-команды echo. Добавьте в файл плейбука следующее содержимое:

```
- name: Echo Playbook
  hosts: all
  tasks:
  - name: Execute command
    shell: echo "My hostname is "$HOSTNAME
    register: output
  - name: Display echo
    debug:
      var: output.stdout_lines
```

Запустить выполнение плейбука можно следующей командой:

ansible-playbook playbook.yml

Также, как и при использовании команды ansible, при запуске плейбука можно указать собственные файлы инвентаризации:

ansible-playbook playbook.yml -i /путь/к_файлу/инвентаризации

Для того чтобы узнать, какие узлы затронет выполнение плейбука, следует использовать команду:

ansible-playbook playbooks.yml --list-host

Результат вывода выполнения плейбука разделяется на несколько частей. Рассмотрим каждую из них:

PLAY [Echo Playbook]

В данной строке Ansible сообщает нам, что выполняется плейбук «Echo Playbook». Примечательно: если не добавлять строку с названием плейбука, Ansible вместо названия плейбука в квадратных скобках укажет группу управляемых узлов, на которую распространяется выполнение плейбука.

TASK [Gathering Facts]

ok: [192.168.0.100]

Перед каждым воспроизведением плейбука Ansible автоматически собирает информацию об используемом окружении на управляемом узле. Например, используемую версию Python. В этой строке Ansible оповещает о том, что данные успешно собраны.

```
TASK [Execute Command]
changed: [192.168.0.100]
TASK [Display echo]
ok: [192.168.0.100] => {
    "output.stdout_lines": [
        "My hostname is localhost.localdomain"
    ]
}
```

Следующие сообщения уже говорят о статусе выполнения указанных в плейбуке задач. Первое сообщение указывает на то, что на управляемом узле были выполнены изменения. Однако в действительности никаких изменений не производилось. Такая ситуация возникает из-за того, что Ansible интерпретирует выполнение определённой команды или задачи как потенциальное изменение состояния системы.

Второе сообщение уже выводит желаемый результат выполнения команды echo.

PLAY RECAP

192.168.0.100 : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

Последнее сообщение представляет собой сводку выполнения плейбука, содержащую итоговую информацию о количестве выполненных задач, а также о количестве изменений, внесённых на управляемом узле. В этом отчёте также указывается количество задач, которые не были выполнены или завершились с ошибкой.

Разберём код подробнее. В первой строке используется ключевое слово name для указания названия плейбука:

```
- name: Echo Playbook
```

Указание имён необязательно, но они улучшают читаемость кода и помогают ориентироваться в логике работы плейбука. Допускается использование названий на русском языке.

В следующей строке указывается группа управляемых узлов:

```
hosts: all
```

Так как в данном примере стоит цель выполнения плейбука на всех управляемых узлах, то в значении этой строки указывается ключевое слово all.

Со строки tasks: начинается перечисление задач, которые должны быть выполнены на управляемых узлах. Каждой задаче назначено определённое имя аналогично имени плейбука. Рассмотрим код первой задачи:

```
- name: Execute command
  shell: echo "My hostname is "$HOSTNAME
  register: output
```

Для выполнения bash-команды используется встроенный в Ansible модуль shell. Именно работу этого модуля Ansible интерпретирует как изменение на управляемом узле. Вывод выполненной команды передаётся в переменную output с помощью ключевого слова register.

Результат команды echo можно получить из значения свойства переменной output под названием stdout_lines. Для вывода результата используется встроенный модуль debug:

```
- name: Display echo
  debug:
    var: output.stdout_lines
```

Следует отметить, что соблюдение правильного количества отступов при написании сценариев на YAML является критически важным для корректной работы сценариев.

Для некоторых операций на управляемых узлах требуются привилегии администратора.

В качестве примера рассмотрим создание плейбука для установки программ на управляемые узлы. Для этой задачи плейбук должен выглядеть следующим образом:

```
- hosts: all
  become: yes
  tasks:
  - name: Install programm
    vars:
      packages:
      - unrar
      - p7zip
    dnf:
      name: "{{ packages }}"
```

В результате выполнения плейбука на управляемые узлы будут установлены пакеты unrar и p7zip. Для запуска плейбука потребуется использовать ключ -K (потребуется ввести пароль администратора управляемого узла):

```
ansible-playbook playbook.yml -K
```

При успешном выполнении задачи Ansible может вывести в терминал следующее сообщение:

```
TASK [Install programm]
changed: [192.168.0.100]
```

Важным нововведением в коде плейбука является строка:

```
become: yes
```

Эта строка указывает плейбуку использовать привилегии администратора на управляемых узлах. Вместо добавления этой строки в сценарий можно добавить ключ -b в команду запуска плейбука:

```
ansible-playbook playbook.yml -K -b
```

В коде задачи создаётся массив с именем packages, в котором перечисляются пакеты, которые будут установлены на управляемых узлах.

```
vars:
  packages:
  - unrar
  - p7zip
```

Для установки пакетов на узлах под управлением РЕД ОС используется встроенный модуль dnf. Важно понимать, что параметр

name в данном контексте не является названием задачи, а представляет собой свойство модуля, в которое передаются названия пакетов, подлежащих установке. Именно в это свойство необходимо передать массив packages для установки необходимых пакетов.

```
dnf:
  name: "{{ packages }}"
```

По умолчанию при запуске плейбука Ansible автоматически проверяет его синтаксис. Однако часто возникает необходимость проверки синтаксиса без запуска плейбука. Для этого можно использовать следующую команду:

```
ansible-playbook --syntax-check your_playbook.yml
```

Пакеты Ansible версии 6.X располагаются в подключаемом репозитории, поэтому порядок установки несколько отличается от обычного.

Установка пакетов Ansible версии 6.X на РЕД ОС 7.3 производится с помощью следующей цепочки команд (потребуются привилегии администратора):

```
dnf install ansible6-release
dnf clean all
dnf makecache
dnf install ansible
```

Задание

1. Создать три виртуальные машины, установить на них операционную систему Ред ОС и наладить между ними внутреннюю виртуальную сеть.
2. Установить на одну из машин систему Ansible, добавить IP-адреса всех трёх машин в файл hosts и установить подключение к ним. Объединить две другие машины в группу под названием [Ваша фамилия на латинице].
3. Создать и выполнить плейбук автоматической установки заданного преподавателем пакета на подключенные узлы.

Контрольные вопросы

1. Что такое Ansible?
2. Что такое плейбук?
3. Как устанавливается подключение к управляемым узлам?
4. Как объединить узлы в группу?
5. Что такое файл инвентаризации?

Контейнеризация в Linux

Для работы понадобится виртуальная машина с установленной операционной системой Ред ОС.

Программное обеспечение 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 установки средства контейнеризации необходимо выполнить команду (потребуются права администратора):

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.

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>

Проверить назначенные права можно, выполнив загрузку тестового образа с правами обычного пользователя:

docker info

При корректной настройке будет получен соответствующий ответ от сервиса Docker.

Часто используемые команды (аргументы команд для краткости не указаны):

Команда	Краткое описание
docker attach	Подключить стандартные каналы ввода-вывода (stdin, stdout, stderr) к активному контейнеру.
docker build	Построить Docker-образ из Docker-файла.
docker builder	Управление модулем построения Docker-образов.
docker commit	Создание нового Docker-образа из активного контейнера.
docker config	Управление конфигурациями Docker.
docker container	Управление контейнерами: • docker container create - создать новый контейнер; • docker container exec - выполнить команду в активном контейнере; • docker container run - выполнить команду в новом контейнере; • docker container start - активировать контейнер (контейнеры); • docker container stop - деактивация контейнера.
docker context	Управление контекстами Docker.
docker cp	Копирование файлов или каталогов между локальной файловой системой и файловой системой контейнера.
docker create	Создать новый изменяемый слой в указанном контейнере.
docker diff	Вывести список файлов и каталогов, изменённых с момента создания контейнера.
docker events	Вывести событий, произошедших с различными объектами Docker.
docker exec	Выполнить команду в активном контейнере.
docker export	Экспортировать файловую систему контейнера как архив формата tar.
docker history	Показать историю образа
docker image	Управление образами: • docker image build - создание нового образа.
docker images	Вывести список образов верхнего уровня.
docker import	Создать файловую систему образа из архива.

docker info	Вывод системной информации.
docker inspect	Вывод подробной информации об объектах Docker.
docker kill	Принудительно деактивировать активный контейнер (контейнеры).
docker load	Загрузить образ из архива tar или из стандартного ввода.
docker login	Войти в реестр образов.
docker logout	Выйти из реестра образов.
docker logs	Извлечь журналы контейнера.
docker manifest	Управление манифестами и списками манифестов Docker.
docker network	Управление сетями.
docker node	Управление узлами кластеров Docker.
docker pause	Приостановить все процессы в активном контейнере (контейнерах).
docker plugin	Управление плагинами.
docker port	Вывести список отображения портов контейнера.
docker ps	Вывести список активных контейнеров.
docker pull	Загрузить образ на локальный компьютер из реестра образов.
docker push	Загрузить образ с локального компьютера в реестр образов.
docker rename	Переименовать контейнер.
docker restart	Перезапустить контейнер (контейнеры).
docker rm	Удалить контейнер (контейнеры).
docker rmi	Удалить образ (образы).
docker run	Выполнить команду в новом контейнере, то есть: создать в существующем образе новый изменяемый слой, и выполнить команду, сохраняя изменения в этом слое. • docker run --rm - удалить новый (изменённый) слой после деактивации контейнера, т.е. сохранить образ неизменным.
docker save	Сохранить образ (образы) в архиве tar (через стандартный вывод по умолчанию).
docker search	Поиск образов Docker в сети Интернет.
docker secret	Управление паролями кластеров Docker.
docker service	Управление сервисами кластеров Docker.
docker stack	Управление стеками.
docker start	Запустить контейнер (контейнеры).
docker stats	Отобразить в режиме реального времени статистику потребления ресурсов контейнером.
docker stop	Остановить активный контейнер (контейнеры).
docker swarm	Управление кластерами Docker.
docker system	Управление службой Docker.
docker tag	Создать тег (метку) образа, ссылающийся на существующий образ.

docker top	Вывести список процессов активного контейнера.
docker trust	Управление ключами и подписями образов.
docker unpause	Продолжить выполнение приостановленного активного контейнера (контейнеров).
docker update	Обновить конфигурацию контейнера (контейнеров).
docker version	Отобразить версию Docker.
docker volume	Управление томами хранения данных для контейнеров.
docker wait	Ожидание завершения работы контейнера (контейнеров) и вывод кодов завершения.

При возникновении проблем с обращением к образам из общедоступного хранилища Docker Hub для обеспечения возможности продолжения корректной работы с образами рекомендуется настроить подготовленные зеркала.

Для настройки обращения docker к определенному зеркалу необходимо выполнить следующий алгоритм действий:

1а. В файл /etc/docker/daemon.json добавить строку вида:

"registry-mirrors": ["<адрес_зеркала>"]

где <адрес_зеркала> – адрес настроенного зеркала хранилища образов.

Например:

"registry-mirrors": ["https://mirror.gcr.io/"]

где https://mirror.gcr.io/ – публичное зеркало хранилища образов Docker Hub от компании Google.

1б. Если файл /etc/docker/daemon.json пуст или не существует, внести строки вида:

{

"registry-mirrors": ["<адрес_зеркала>"]

}

где <адрес_зеркала> – адрес настроенного зеркала хранилища образов.

Например:

{

"registry-mirrors": ["https://mirror.gcr.io/"]

}

где https://mirror.gcr.io/ – публичное зеркало хранилища образов Docker Hub от компании Google.

2. Перезапустить службу docker:

systemctl daemon-reload

systemctl restart docker

На пакетной базе РЕД ОС разработан ряд docker-образов и создан собственный реестр docker-образов – registry.red-soft.ru.

В зависимости от содержащихся образов реестр имеет следующие ветки:

ubi7 – содержит базовые и прикладные образы, построенные на пакетах из репозитория Стандартной редакции РЕД ОС 7.3;

k8s7 – содержит образы кластера Kubernetes, построенные на пакетах из репозитория Стандартной редакции РЕД ОС 7.3;

redos7c – содержит базовые образы, построенные на пакетах из репозитория Сертифицированной редакции РЕД ОС 7.3;

k8s7c – содержит образы кластера Kubernetes, построенные на пакетах из репозитория Сертифицированной редакции РЕД ОС 7.3.

Для получения списка доступных образов в реестре необходимо выполнить команду (в примере будет выведен список образов ветки ubi7):

docker search registry.red-soft.ru/ubi7

Для установки требуемого образа из реестра необходимо выполнить команду:

docker pull registry.red-soft.ru/ubi7/<название_образа>

Универсальные базовые образы (UBI) — это легковесная и безопасная основа для создания облачных и веб-приложений в контейнерах. На основе UBI можно создавать свои контейнерные приложения и собственные образы.

В реестре образов РЕД ОС на данный момент доступны следующие базовые образы:

ubi – "урезанный образ", который использует dnf в качестве менеджера пакетов;

ubi-minimal – еще более "урезанный" образ, который использует microdnf в качестве менеджера пакетов;

ubi-micro – максимально "урезанный" образ, в котором отсутствует собственный менеджер пакетов. Данный docker-образ предназначен для запуска статически собранных бинарных файлов.

Также в реестре образов доступны образы типа S2I (Source-to-Image).

Source-to-Image — это процесс сборки для создания воспроизводимых образов контейнеров из исходного кода. Source-to-Image создает готовые к использованию образы на основе исходных кодов приложений, которые, в свою очередь, могут запускаться в виде собранного приложения.

В реестре образов РЕД ОС доступны следующие образы S2I:

s2i-core — предоставляет образам инструменты, необходимые для использования функциональности source-to-image, сохраняя при этом минимальный размер образа;

s2i-base — основан на s2i-core, предоставляет образам инструменты, необходимые для использования функциональности source-to-image, а также содержит различные библиотеки, которые служат основой для создания других s2i-образов — таких как s2i-python, s2i-ruby, s2i-nodejs и прочих.

В централизованном хранилище хранятся доступные docker-образы.

Одним из примеров общедоступных хранилищ образов является Docker Hub, по умолчанию Docker настроен на поиск образов в нем. Также есть возможность настройки своего собственного хранилища.

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

Установка и настройка локального реестра docker-образов из репозитория РЕД ОС

Для установки локального реестра docker-образов из репозитория РЕД ОС выполните команду:

dnf install registry

Запустите службу registry:

systemctl enable registry.service --now

Убедитесь, что служба активна, выполнив:

systemctl status registry.service

В статусе должно быть указано active(running).

Важно!

Необходимо настроить демон docker на обращение к приватному реестру образов. Для этого добавьте в файл /etc/docker/daemon.json следующую строку:

"insecure-registries": ["127.0.0.1:5000"],

Локальный реестр docker-образов готов к использованию.

Для настройки локального хранилища образов путем использования контейнеров в реестре docker-образов РЕД СОФТ предоставлен образ registry.red-soft.ru/ubi7/registry, являющийся реализацией инструмента хранения и обмена образами Docker Registry.

Для создания локального хранилища образов запустите контейнер на основе образа registry.red-soft.ru/ubi7/registry командой:

docker run -d -p 5000:5000 --restart=always --name test_reg_2 registry.red-soft.ru/ubi7/registry:latest

где:

-p 5000:5000 — порт для обращения к реестру по умолчанию;

--restart=always — автоматический запуск контейнера после перезапуска хостовой ОС;

--name test_reg_2 — имя контейнера;

registry.red-soft.ru/ubi7/registry:latest — необходимый образ.

Локальный реестр образов готов к использованию.

Локальное хранилище образов является контейнером и хранит образы внутри себя в каталоге /var/lib/registry. Однако при переустановке контейнера все данные внутри него будут уничтожены. Для избежания потери данных к контейнеру необходимо подключить внешнее хранилище данных.

Для этого на хостовой ОС сначала следует создать каталог, предназначенный для хранения образов:

mkdir /reg_images

Далее необходимо остановить и уничтожить имеющийся контейнер с локальным хранилищем:

docker container stop test_reg_2 && docker container rm -v test_reg_2

После этого следует запустить контейнер снова с дополнительным ключом, который отвечает за подключение к нему ранее созданного каталога:

docker run -d -p 5000:5000 --restart=always --name test_reg_2 -v /reg_images:/var/lib/registry registry.red-soft.ru/ubi7/registry:latest

где:

-p 5000:5000 — порт для обращения к реестру по умолчанию;

--restart=always — автоматический запуск контейнера после перезапуска хостовой ОС;

--name test_reg_2 — имя контейнера;

-v /reg_images:/var/lib/registry — подключение каталога /reg_images к контейнеру в качестве каталога /var/lib/registry;

registry.red-soft.ru/ubi7/registry:latest — необходимый образ.

Для проверки корректности настройки локального хранилища образов в качестве примера будет загружен образ nodejs-18 из реестра образов РЕД СОФТ registry.red-soft.ru/ubi7:

docker pull registry.red-soft.ru/ubi7/nodejs-18

Далее будет добавлен тег к загруженному образу:

docker tag registry.red-soft.ru/ubi7/nodejs-18 localhost:5000/js18_test:latest

Затем образ загружается в локальное хранилище:

docker push localhost:5000/js18_test:latest

The push refers to repository [localhost:5000/js18_test]
0932ece797fa: Pushed
6512634988ef: Pushed
526374a3d4d5: Pushed
latest: digest: sha256:fe07d0235d269961ba25f5853809ff81807f52d0aa75a77e51899a540975a675 size: 954

Проверка наличия загруженного образа в созданном каталоге /reg_images:

ls /reg_images/docker/registry/v2/repositories/
js18_test

Локальное хранилище с подключенным внешним каталогом успешно работает.

Бывают ситуации, когда контейнеры могут быть остановлены вследствие определенных факторов. Для того чтобы остановленные контейнеры не приходилось запускать вручную, можно использовать настройку их автоматического запуска.

Далее будет рассмотрен пример настройки автоматического запуска локального реестра registry, настроенного как контейнер с именем test_reg_systemd и внешним каталогом для хранения образов /reg_images_sys.

Примечание.

Настройка автоматического запуска контейнера через службу systemd актуальна только в тех случаях, когда при запуске контейнера командой docker run не была использована опция --restart=always.

Предварительно был создан внешний каталог для хранения образов /reg_images_sys и запущен контейнер с локальным хранилищем следующими командами:

mkdir /reg_images_sys

docker run -d -p 5000:5000 --name test_reg_systemd -v /reg_images_sys:/var/lib/registry registry.red-soft.ru/ubi7/registry:latest

После запуска контейнера registry необходимо создать файл настройки для службы systemd со следующим содержимым:

nano /etc/systemd/system/registry.service

[Unit]

Description=registry container

Requires=docker.service

After=docker.service

[Service]

Restart=always

ExecStart=/usr/bin/docker start -a test_reg_systemd

ExecStop=/usr/bin/docker stop -t 15 test_reg_systemd

TimeoutSec=30

[Install]

WantedBy=multi-user.target

Сохраните изменения в файле и закройте его.

После этого необходимо запустить настроенную службу и добавить ее в автозагрузку командой:

systemctl enable registry.service --now

Проверьте статус службы:

systemctl status registry.service

В статусе должно быть указано active(running).

После перезагрузки хостовой ОС проверьте статус контейнера командой:

docker ps -a

В Linux также имеется инструмент с графическим интерфейсом Portainer для управления контейнерами Docker, K8s, Podman и другими платформами. Он позволяет создавать, запускать, останавливать, перезапускать и удалять контейнеры через веб-интерфейс.

Установка Portainer доступна как из репозитория РЕД ОС, так и с помощью docker путем создания контейнера portainer.

Для установки portainer из репозитория РЕД ОС выполните команду (потребуются права администратора):

dnf install portainer-ce

Запустите службу portainer:

systemctl enable portainer.service --now

Убедитесь, что служба активна, выполнив:

systemctl status portainer.service

В статусе должно быть указано active(running).

Важно!

Для корректной работы в Portainer должен быть установлен и активен docker.

Получить доступ к Portainer можно через веб-интерфейс на порту 9443. Для этого в адресной строке браузера пропишите https://localhost:9443.

Будет открыта страница с веб-интерфейсом portainer и формой для создания пользователя, под которым будет производиться вход в систему.

Для развертывания контейнера с Portainer необходимо сначала создать хранилище данных командой:

docker volume create portainer_data

Затем для установки и запуска контейнера следует выполнить команду:

docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/var/lib/portainer/data registry.red-soft.ru/ubi7/portainer

Получить доступ к Portainer можно через веб-интерфейс на порту 9000. Для этого в адресной строке браузера пропишите https://localhost:9000. Будет открыта страница с веб-интерфейсом portainer.

Задание

1. Установить docker и настроить локальный реестр образов.
2. Установить Portainer и получить доступ к контейнерам.

3. Найти в DockerHub официальный образ веб-сервера nginx и скачать его.
4. Проверить работу веб-сервера, а также выполнение команд внутри работающего контейнера.

Контрольные вопросы

1. Что такое docker?
2. В чем ключевое отличие контейнера от виртуальной машины?
3. Каково основное преимущество использования Docker в rootless режиме?
4. Как посмотреть журнал контейнера, запущенного в фоновом режиме?
5. В чем разница между командами docker stop и docker rm?

Практические занятия

Все практические занятия проводятся под прямым руководством преподавателя, выполнение практических примеров вне контакта с преподавателем не предусмотрено. По итогам занятий проводятся устные опросы для проверки усвоения материала.

1. Введение в системное администрирование и DevOps

Цель практической работы: на практических примерах дать студентам дать общее представление о программных компонентах ОС Linux, особенностях архитектуры систем на базе данного ядра, топологии, уровнях надежности, а также о использовании системно доступной документации (мануалах).

Описание работы

Рассматриваются практические примеры основных программных компонентов Linux, их вызовы, особенности архитектуры, топологии, уровни ограничения доступа, перспективы Linux.

2. Основы администрирования ОС Linux

Цель практической работы: рассмотреть на практике структуру вычислительных систем, основные права доступа администратора и пользователя в файловой системе Linux.

Описание работы

Рассматриваются практические примеры применения программных продуктов ОС на базе ядра Linux в целях просмотра и модификации прав доступа пользователей к файлам в файловой системе. Рассматриваются основные права доступа к файлу, права пользователя, пользователей, входящих в группу, а также пользователей, не относящихся к перечисленным ранее. Объясняются особенности работы с файлами в режиме суперпользователя.

3. Подходы к распределению задач в рамках единой среды исполнения. Использование прослоек абстракции от оборудования (HAL) при постановке задач. Особенности управления доступом пользователей к конкретным узлам в Slurm. Энергосбережение

Цель практической работы: на практических примерах дать студентам общее представление о видах автоматического распределения задач в вычислительном комплексе.

Описание работы

Рассматриваются на практике основные виды программного обеспечения систем постановки задач в очередь PBS, Torque, Slurm, особенности настройки и использования.

4. Особенности настройки компонентов мониторинга Grafana. Использование программного обеспечения на языке python/bash для создания собственных метрик

Цель практической работы: на практических примерах дать студентам общее представление о создании метрик мониторинга в системе Grafana.

Описание работы

Рассматриваются практические примеры задач и методов инженерии знаний, систем, основанных на знаниях. Дается обзор задач и методов систем бизнес-аналитики. Рассматриваются форматы представления 9 данных и инструментальные средства их обработки и преобразования. Рассматривается моделирование бизнес-аналитики с помощью нотации языка UML.

Литература

1. Мошков, М. Е. Введение в системное администрирование Unix : учебное пособие / М. Е. Мошков. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2025. — 207 с. – Текст : электронный // Цифровой образовательный ресурс IPR SMART. — URL: https://www.iprbookshop.ru/146338.html (дата обращения: 08.02.2026).
2. Гончарук, С. В. Администрирование ОС Linux : учебное пособие / С. В. Гончарук. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2024. — 163 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART. — URL: https://www.iprbookshop.ru/133916.html (дата обращения: 08.02.2026).
3. Шлаев, Д.В. Инновационные подходы к визуализации и разработке с применением унифицированного языка моделирования (UML) : учебное пособие / Д. В. Шлаев, А. А. Сорокин, С. В. Аникуев, Ю. В. Орел. — Ставрополь : АГРУС, 2024. — 72 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART. — URL: https://www.iprbookshop.ru/156601.html (дата обращения: 08.02.2026).