Linux, Docker, Bash, окружение для разработки

**Оглавление**

[**Цель работы 2**](#_Toc42428338)

[Краткие теоретические сведения 2](#_Toc42428339)

[**2 Ход работы 8**](#_Toc42428340)

[2.1 Знакомство с терминалом 8](#_Toc42428341)

[2.3 Подготовка к установке Docker 10](#_Toc42428342)

[2.4 Установка Docker 16](#_Toc42428343)

[2.5 Использование команды Docker без sudo (опционально) 17](#_Toc42428344)

[2.6 Использование команды Docker 18](#_Toc42428345)

[2.7 Запуск контейнера Docker 22](#_Toc42428346)

[**2.7.1 Установка и работа с Nano 25**](#_Toc42428347)

[**2.7.2 Создание и работа с Dockerfile 30**](#_Toc42428348)

[2.8 Установка Docker для Windows 34](#_Toc42428349)

[**3 Варианты заданий 39**](#_Toc42428350)

[**4 Справочник по командам BASH 51**](#_Toc42428351)

[**5 Контрольные вопросы 69**](#_Toc42428352)

[**6 Список литературы 70**](#_Toc42428353)

# Цель работы

Целью лабораторной работы является ознакомление с операционной системой и основными командами ОС Unix/Linux, возможностями Docker для контейнеризации окружения программного обеспечения и его зависимостей, средств разработки и сборки ПО. Необходимо подготовить Dockerfile, в котором осуществляется виртуализация операционной системы и устанавливаются необходимые пакеты.

В ходе лабораторной работы требуется выполнить задание и разработать скрипт, соответствующее вашему варианту.

После выполнения задания, необходимо загрузить код готовой программы и Dockerfile на github.com.

**Краткие теоретические сведения**

***Unix*** - семейство [операционных систем](https://www.dmosk.ru/terminus.php?object=os). Благодаря высокой производительности и стабильности, активно используются для серверов и серверного оборудования. Аббревиатура UNIX расшифровывается как Uniplexed Information and Computing System.

Основными отличительными особенностями являются:

* Многозадачность;
* многопользовательский режим;
* поддержка большого спектра различного оборудования;
* использование командной строки как основного инструмента управления.

***Linux*** — это операционная система, ядро которой распространяется на бесплатной основе. Она состоит из ядра системы и набора небольших программ, взаимодействующих с этим ядром. На основе ядра можно собрать собственную версию операционной системы, дистрибутив. На сегодняшний день дистрибутивов Linux существует не менее 9000, новые появляются регулярно.

Почти все современные ОС основаны на UNIX: MacOS, Android, Linux, BSD и т.д. Конкретно дистрибутивами операционной системы Linux сегодня активно пользуются:

* Профессиональны веб-программисты;
* программисты-любители;
* любые пользователи, компьютерные потребности которых исчерпываются сидением в интернете и работой с рядом основных программ;
* пользователи, которым нравится настраивать операционную систему под себя, конфигурировать и подгонять;
* пользователи, которые устали от проблем с Windows и хотят пользоваться простой системой, защищённой от вирусов.

***Docker*** — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой [контейнеризации](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F). Позволяет «упаковать» приложение со всем его [окружением](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5&action=edit&redlink=1) и зависимостями в контейнер, который может быть перенесён на любую [Linux](https://ru.wikipedia.org/wiki/Linux)-систему с поддержкой [cgroups](https://ru.wikipedia.org/wiki/Cgroups) в [ядре](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux), а также предоставляет среду по управлению контейнерами. [Это ПО с открытым кодом](https://www.docker.com/), принцип работы которого основан на контейнерах. Когда вы разрабатываете приложение, вам нужно предоставить код вместе со всеми его составляющими, такими как библиотеки, сервер, базы данных и т. д. Вы можете оказаться в ситуации, когда приложение работает на вашем компьютере, но отказывается включаться на устройстве другого пользователя. Эта проблема решается через создание независимости ПО от системы. Докер же просто разделяет ядро ОС на все контейнеры (Docker container), работающие как отдельные процессы. Это не единственная подобная платформа, но, бесспорно, одна из самых популярных и востребованных.

Как работает Docker? Работа Docker основана на принципах клиент-серверной архитектуры, которая основана на взаимодействии клиента с веб-сервером (хостом). Первый отправляет запросы на получение данных, а второй их предоставляет (рисунок 1.1).

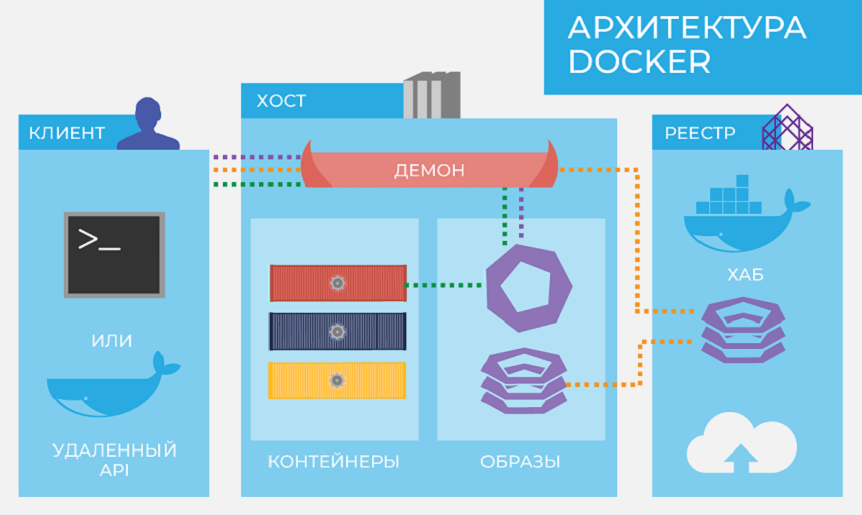


Рисунок 1.1 – Принцип работы Docker

Пользователь отдает команду с помощью клиентского интерфейса Docker-демону, развернутому на Docker-хосте. Например, скачать готовый образ из реестра (хранилища Docker-образов). Взаимодействие между клиентом и демоном обеспечивает REST API. Демон может использовать публичный (Docker Hub) или частный реестры.

Исходя из команды, заданной клиентом, демон выполняет различные операции с образами на основе инструкций, прописанных в файле Dockerfile (о нем позже). Например, производит их автоматическую сборку.

Работа образа в контейнере. Например, запуск docker-image или удаление контейнера.

Предшественниками контейнеров Docker были виртуальные машины. Виртуальная машина, как и контейнер, изолирует от внешней среды приложение и его зависимости. Однако контейнеры Docker обладают преимуществами перед виртуальными машинами. Так, они потребляют меньше ресурсов, их очень легко переносить, они быстрее запускаются и приходят в работоспособное состояние.

Виртуальные машины имеют хост-ОС и гостевую ОС внутри каждой виртуальной машины. Гостевой ОС может быть любая ОС, например, Linux или Windows, независимо от хост-ОС. Контейнеры Docker, напротив, размещаются на одном физическом сервере с операционной системой хоста, которая разделяет их между собой. Совместное использование ОС хоста между контейнерами делает их легкими и увеличивает время загрузки. Docker-контейнеры считаются подходящими для запуска нескольких приложений на одном ядре ОС; тогда как виртуальные машины необходимы, если приложения или службы требуются для работы в разных ОС.

Виртуальные машины автономны со своим ядром и функциями безопасности. Поэтому приложения, которым требуются дополнительные привилегии и безопасность, работают на виртуальных машинах.

С другой стороны, предоставление корневого доступа к приложениям и запуск их из административных помещений не рекомендуется в случае контейнеров Docker, поскольку контейнеры совместно используют ядро ​​хоста. Контейнерная технология имеет доступ к подсистемам ядра; в результате одно зараженное приложение способно взломать всю хост-систему.

Виртуальные машины изолированы от их ОС, поэтому они не переносятся на несколько платформ без проблем с совместимостью. На уровне разработки, если приложение должно тестироваться на разных платформах, необходимо учитывать контейнеры Docker.

Пакеты Docker-контейнеров являются автономными и могут запускать приложения в любой среде, и, поскольку им не требуется гостевая ОС, их можно легко перенести на разные платформы. Контейнеры Docker могут быть легко развернуты на серверах, поскольку легковесные контейнеры можно запускать и останавливать за очень короткое время по сравнению с виртуальными машинами.

Виртуальные машины более ресурсоемки, чем контейнеры Docker, поскольку виртуальные машины должны загружать всю ОС для запуска. Облегченная архитектура контейнеров Docker менее ресурсоемка, чем виртуальные машины.

В случае виртуальных машин такие ресурсы, как ЦП, память и ввод-вывод, могут не выделяться для контейнеров постоянно – в отличие от контейнеров, где использование ресурсов связано с нагрузкой или трафиком.

Простое и удобное масштабирование и дублирование контейнеров по сравнению с виртуальными машинами, поскольку в них нет необходимости устанавливать операционную систему.

Сравним Docker с самым обычным контейнером, не важно каким именно – пластиковым для кухонных нужд, большим железным контейнером для дальних перевозок или же контейнер для инструментов. Все они служат для хранения, перевозки чего-либо или использования содержимого. Как и все до этого перечисленные контейнеры, контейнер Docker обладает следующими характеристиками:

* В нём можно что-то хранить;
* его можно переносить. Контейнер Docker можно использовать на локальном компьютере, на компьютере коллеги, на сервере поставщика облачных услуг. Это роднит контейнеры Docker с обычными контейнерами, в которых, например, перевозят разные милые сердцу безделушки при переезде в новый дом;
* в контейнер удобно что-то класть и удобно что-то из него вынимать;
* если вам нужен контейнер, его можно купить в магазине или же заказать в интернет-магазине. В этот магазин контейнеры попадают от производителей, которые делают их в огромных количествах, используя пресс-формы. В случае с контейнерами Docker то, что можно сравнить с пресс-формой, а именно — образ контейнера, хранится в специальном репозитории. Если вам нужен некий контейнер, вы можете загрузить из репозитория соответствующий образ, и, используя его, этот контейнер создать.

***Docker-daemon*** — сервер контейнеров, входящий в состав программных средств Docker. Демон управляет Docker-объектами (сети, хранилища, образы и контейнеры). Демон также может связываться с другими демонами для управления сервисами Docker.

***Docker-client*** — интерфейс взаимодействия пользователя с Docker-демоном. Клиент и Демон – важнейшие компоненты «движка» Докера (Docker Engine). Клиент Docker может взаимодействовать с несколькими демонами.

***Docker-image*** (образ)— файл, включающий зависимости, сведения, конфигурацию для дальнейшего развертывания и инициализации контейнера. Образы контейнеров Docker можно сравнить с чертежами. Образы — это неизменные шаблоны, которые используются для создания одинаковых контейнеров. В образе контейнера Docker содержится образ базовой операционной системы, код приложения, библиотеки, от которого оно зависит. Всё это скомпоновано в виде единой сущности, на основе которой можно создать контейнер.

***Dockerfile***— описание правил по сборке образа, в котором первая строка указывает на базовый образ. Последующие команды выполняют копирование файлов и установку программ для создания определенной среды для разработки.

***BASH*** — Bourne-Again SHell, самый популярный командный интерпретатор в юниксоподобных системах, в особенности в GNU/Linux. Представляет собой командный процессор, работающий, как правило, в интерактивном режиме в текстовом окне. Bash также может читать команды из файла, который называется [скриптом](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82) (или сценарием). Как и все Unix-оболочки, он поддерживает автодополнение имён файлов и каталогов, подстановку вывода результата команд, переменные, контроль над порядком выполнения, операторы ветвления и цикла. Ниже будет приведен ряд встроенных базовых команд, которые возможно будут использовать для создания скриптов:

**2 Ход работы**

**2.1 Знакомство с терминалом**

Для начала лабораторной работы рекомендуется реализовывать установку Docker на операционной системе Ubuntu, запустить которую можно на виртуальной машине (рисунок 2.1).

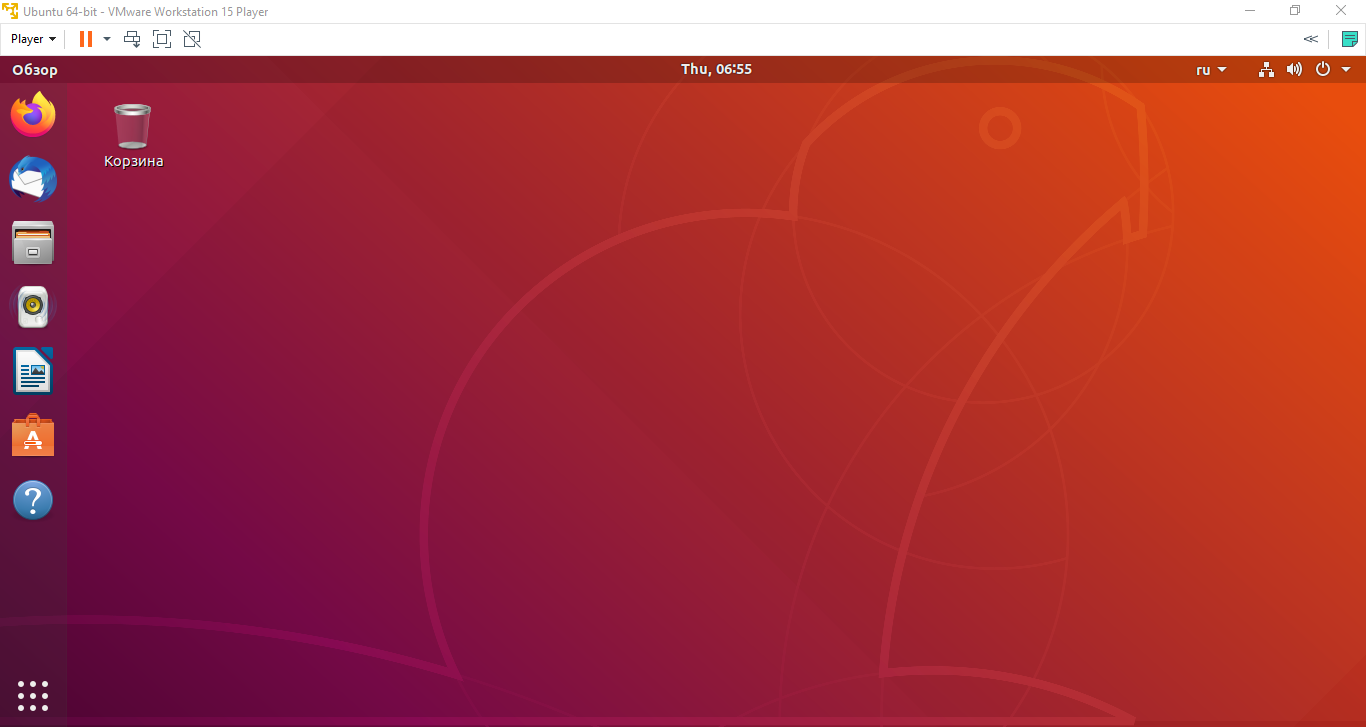


Рисунок 2.1 – Ubuntu реализованная на VMware

После приступим к работе с терминалом. Итак, давайте сначала посмотрим, что из себя представляет этот самый терминал. Найти его, можно набрав «терминал» в строке поиска, или комбинацией клавиш Ctrl+Alt+T. Запустив его вы увидите примерно такое окошечко (рисунок 2.2).

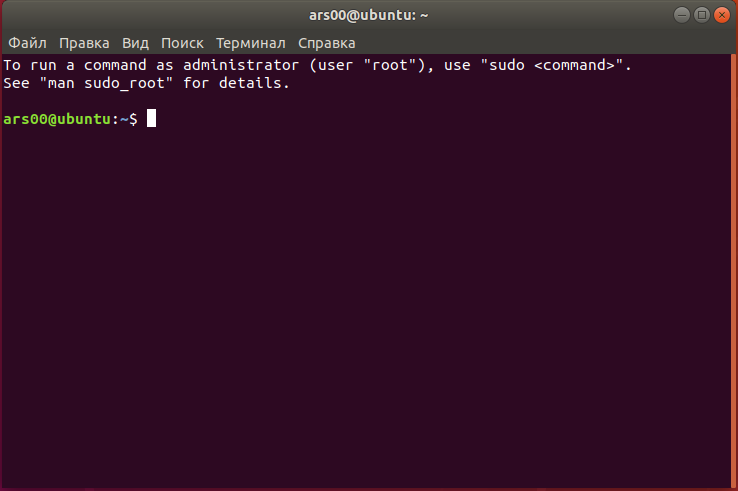


Рисунок 2.2 – Терминал

Знакомьтесь, это и есть терминал. Он создан для того, чтобы выполнять текстовые команды.

Набирать такие команды с клавиатуры посимвольно немного неудобно, поэтому давайте сразу разберёмся с основами управления терминалом. Начнём с копирования/вставки. Стандартные сочетания клавиш Ctrl+C и Ctrl+V в терминале не работают, вместо них используется старая добрая пара Ctrl+Insert с Shift+Insert или же сочетания с Shift: Ctrl+Shift+C для копирования и Ctrl+Shift+V для вставки, как вариант ПКМ -> вставить. Что ж, теперь вы умеете целиком копировать команды из руководств.

Кстати, в большинстве руководств и инструкций вы встретите именно терминальные команды. Это связано с тем, что, графических оболочек очень много, и объяснить, как выполнить какое-то действие для каждой из них бывает очень непросто. А терминал - один для всех, и одна и та же команда работает во всех оболочках (естественно, кроме команд по настройке самой оболочки). К тому же, намного проще дать одну команду, чем объяснить где и как 10 раз нажать мышкой.

Однако, часто всё-таки команды приходится набирать вручную, а не вставлять откуда-то. И вот тут на помощь приходит великолепное свойство терминала, называемое автодополнением. Наберите в терминале символы sudo apt-g, а потом нажмите клавишу Tab. Терминал автоматически дополнит за вас команду. Кстати, sudo apt-get - это основная консольная утилита управления пакетами, но об этом после.

А теперь попробуйте набрать только apt и нажать Tab. Ничего не происходит? А теперь нажмите Tab два раза подряд. Видите, терминал выдал вам список всех команд, начинающихся с apt (рисунок 2.3).

**Команда:** ***sudo apt***

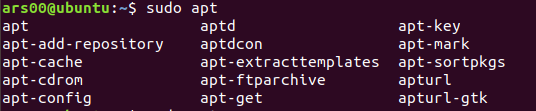


Рисунок 2.3 – Список всех команд начинающихся с apt

**2.3 Подготовка к установке Docker**

Обновите индекс пакетов (загружаем списки пакетов из репозиториев и "обновляем" их, чтобы получить информацию о новейших версиях пакетов и их зависимостях) (рисунок 2.4).

**Команда: *sudo apt-get update***

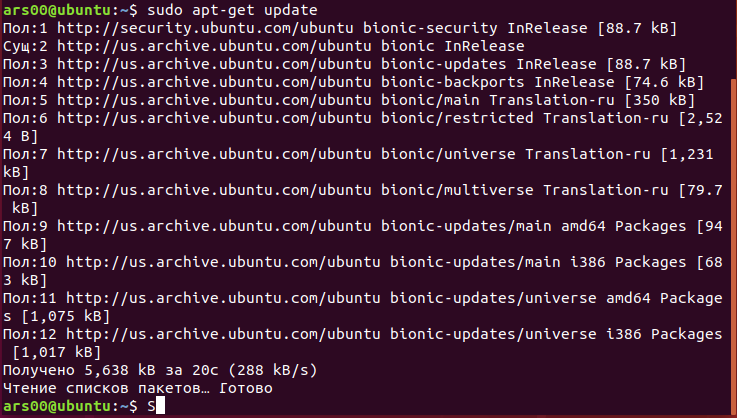


Рисунок 2.4 – Обновление индекс пакетов

Для таких OC как RedHat, CentOS, Scientific Linux (и возможно других) используются немного иные команды, нежели приведенные в примере. В основном задействована команда yum, примеры команд указаны перед контрольными вопросами.

В любой Linux-системе обязательно есть один привилегированный пользователь — root. Этот пользователь имеет права на выполнение любых действий, удаление любых файлов и изменение любых параметров. Как-то ограничить свободу действий root практически невозможно. С другой стороны, все остальные пользователи системы обычно не имеют большинства необходимых прав, например, прав на установку программ, поскольку это является административной операцией, права на которую есть только у root. Ещё одной распространённой операцией, доступной только суперпользователю, является копирование и изменение файлов в системных папках, куда обычный пользователь доступа не имеет.

Раньше данная проблема решалась достаточно просто: при обладании паролем root можно было зайти в систему под его аккаунтом либо временно получить его права, используя команду su. Потом выполнить все необходимые операции и вернуться обратно под обычного пользователя. В принципе, такая схема работает неплохо, однако у неё есть много существенных недостатков, в частности, невозможно никак (точнее, очень сложно) ограничивать административные привилегии только определённым кругом задач.

Поэтому в современных дистрибутивах Linux вместо root аккаунта для администрирования используется утилита sudo.

В Ubuntu по умолчанию root аккаунт вообще отключён, т.е. вы никаким способом не сможете попасть под root, не включив его. root именно что отключён, т.е. он присутствует в системе, под него всего лишь нельзя зайти. Если вы хотите вернуть возможность использовать root, смотрите ниже пункт о включении root аккаунта.

Что такое sudo?

Sudo — это утилита, предоставляющая привилегии root для выполнения административных операций в соответствии со своими настройками. Она позволяет легко контролировать доступ к важным приложениям в системе. По умолчанию, при установке Ubuntu первому пользователю (тому, который создаётся во время установки) предоставляются полные права на использование sudo. Т.е. фактически первый пользователь обладает той же свободой действий, что и root. Однако такое поведение sudo легко изменить, об этом см. ниже в пункте про настройку sudo.

Где используется sudo?

Sudo используется всегда, когда вы запускаете что-то из меню Администрирования системы. Например, при запуске Synaptic вас попросят ввести свой пароль. Synaptic - это программа управления установленным ПО, поэтому для её запуска нужны права администратора, которые вы и получаете через sudo вводя свой пароль.

Однако не все программы, требующие административных привилегий, автоматически запускаются через sudo. Обычно запускать программы с правами администратора приходится вручную.

Что такое apt?

Advanced Packaging Tool – набор утилит для управления программными пакетами в операционных системах основанных на Debian.

Где используется apt?

APT предоставляет дружественную надстройку над DPKG и позволяет:

* устанавливать, удалять и обновлять пакеты;
* решать зависимости;
* искать пакеты по заданным критериям;
* просматривать подробную информацию о пакете;
* манипулировать ключами от репозиториев;
* APT является одной из базовых программ и включена в состав Ubuntu.

Команда apt-get — это мощный консольный инструмент, который работает с Улучшенным инструментарием пакетов (APT) Ubuntu, выполняющий такие функции, как установка новых программных пакетов, обновление имеющихся пакетов, обновления индекса списка пакетов и даже обновление все системы Ubuntu.

Будучи простым консольным инструментом, apt-get имеет ряд преимуществ над другими инструментами управления пакетами, доступными в Ubuntu серверным администраторам. Некоторые из этих преимуществ включают легкое использование простых терминальных соединений (SSH) а также возможность использования в сценариях системных администраторов, которые могут быть автоматизированы с помощью утилиты планирования cron.

Затем установите необходимые пакеты, которые позволяют apt использовать пакеты по HTTPS (рисунок 2.5).

**Команда: *sudo apt install apt-transport-https ca-certificates curl software-properties-common***

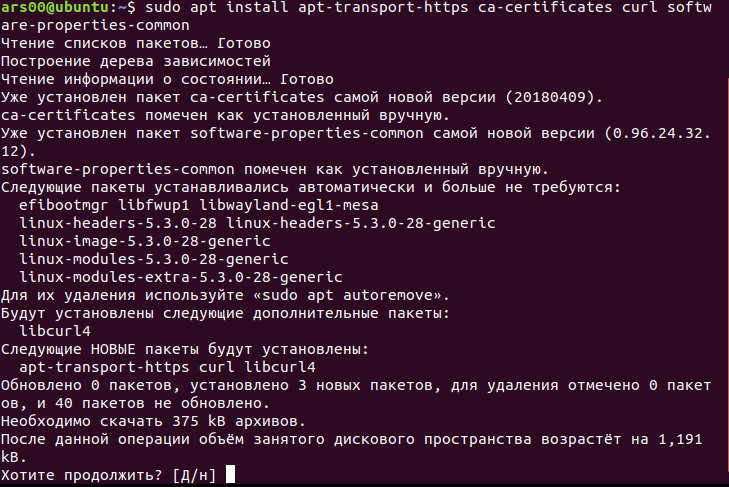


Рисунок 2.5 – Установка необходимых пакетов

Затем добавьте в свою систему ключ GPG официального репозитория Docker (рисунок 2.6).

**Команда: *curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -***



Рисунок 2.6 – Добавление в систему ключа GPG

Curl — это сокращение от “Client URL”. Утилита доступна в большинстве систем на основе Unix и предназначена для проверки подключения к URL-адресам. Кроме того команда Curl — отличный инструмент передачи данных. Давайте же узнаем, как ею пользоваться.

Команда Curl поддерживает следующий список протоколов:

* HTTP и HTTPS;
* FTP и FTPS;
* IMAP и IMAPS;
* POP3 и POP3S;
* SMB и SMBS;
* SFTP;
* SCP;
* TELNET;
* GOPHER;
* LDAP и LDAPS;
* SMTP и SMTPS.

GPG — аббревиатура для [GNU Privacy Guard](https://www.gnupg.org/), реализующего стандарт OpenPGP, который позволяет подписывать и шифровать сообщения и файлы. Это полезно в ряде ситуаций.

Добавьте репозиторий Docker в список источников пакетов APT (рисунок 2.7).

**Команда: *sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"***

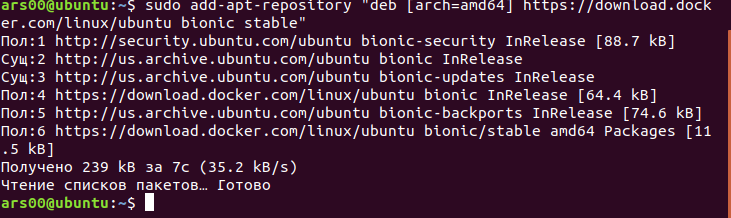


Рисунок 2.7 – Добавление репозитория Docker в список источников пакетов APT

Затем обновите базу данных пакетов информацией о пакетах Docker из вновь добавленного репозитория (рисунок 2.8).

**Команда: *sudo apt-get update***

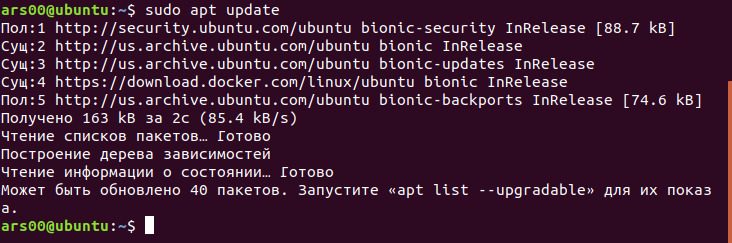


Рисунок 2.8 – Обновление базы данных пакетов

Следует убедиться, что вы устанавливаете Docker из репозитория Docker, а не из репозитория по умолчанию Ubuntu.

**Команда: *apt-cache policy docker-ce***

Вывод получится приблизительно следующий. Номер версии Docker может быть иным (рисунок 2.9)

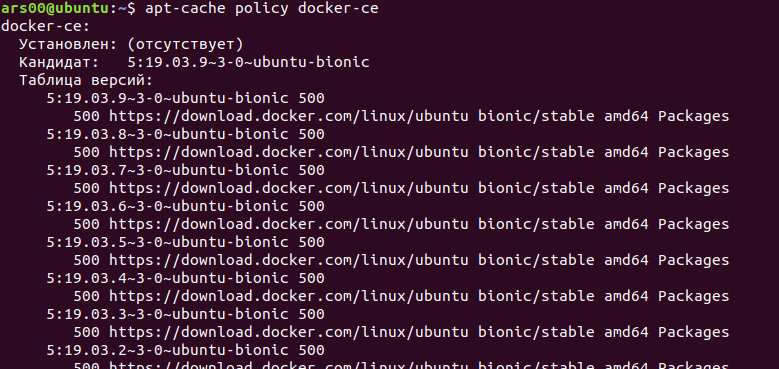


Рисунок 2.9 – Проверка установки

Обратите внимание, что docker-ce не устанавливается, но для установки будет использован репозиторий Docker для Ubuntu 18.04 (bionic).

**2.4 Установка Docker**

Далее установите Docker (рисунок 2.10)

**Команда: *sudo apt install docker-ce***

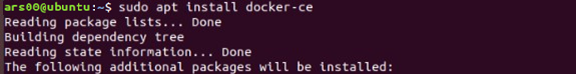


Рисунок 2.10 – Установка docker

Теперь Docker установлен, демон запущен, и процесс будет запускаться при загрузке системы.  Убедитесь, что процесс запущен (рисунок 2.11).

**Команда: *sudo systemctl status docker***

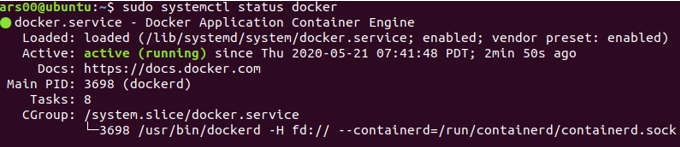


Рисунок 2.11 – Проверка запущен ли процесс

Вывод должен быть похож на представленный ниже, сервис должен быть запущен и активен (рисунок 2.11).

При установке Docker мы получаем не только сервис (демон) Docker, но и утилиту командной строки docker или клиент Docker. Использование утилиты командной строки docker рассмотрено ниже.

**2.5 Использование команды Docker без sudo (опционально)**

По умолчанию, запуск команды docker требует привилегий пользователя root или пользователя группы docker, которая автоматически создается при установке Docker. При попытке запуска команды docker пользователем без привилегий sudo или пользователем, не входящим в группу docker, выводные данные будут выглядеть следующим образом:

docker: Cannot connect to the Docker daemon. Is the docker daemon running on this host?.

See 'docker run --help'.

Чтобы не вводить sudo каждый раз при запуске команды docker, добавьте имя своего пользователя в группу docker (рисунок 2.12).

**Команда: *sudo usermod -aG docker ${USER}***



Рисунок 2.12 – Добавление пользователя в группу docker

Для применения этих изменений в составе группы необходимо разлогиниться и снова залогиниться на сервере или задать следующую команду (рисунок 2.13).

**Команда: *su - ${USER}***



Рисунок 2.13 – Применение изменений через команду

Для продолжения работы необходимо ввести пароль пользователя.

Убедиться, что пользователь добавлен в группу docker можно следующим образом (рисунок 2.14).

**Команда: *id -nG***



Рисунок 2.14 – Проверка добавления пользователя

**2.6 Использование команды Docker**

Команда docker позволяет использовать различные опции, команды с аргументами. Синтаксис выглядит следующим образом:

docker [option] [command] [arguments]

Для просмотра всех доступных подкоманд введите следующую команду (рисунок 2.15).

**Команда: *docker***



Рисунок 2.15 – Команда для вывода листинга подкоманд

Полный список подкоманд Docker (таблица 1).

Таблица 1 - Список подкоманд docker

|  |  |
| --- | --- |
| Команда | Описание |
| attach | подключиться к запущенному контейнер |
| build | собрать образ из инструкций dockerfilе |
| commit | создать новый образ из изменений контейнера |
| cp | копировать файлы между контейнером и файловой системой |
| create | создать новый контейнер |
| diff | проверить файловую систему контейнера |
| events | посмотреть события от контейнера |
| exec | выполнить команду в контейнере |
| export | извлечь содержимое контейнера в архив |
| history | посмотреть историю изменений образа |
| images | список установленных образов |
| import | создать контейнер из архива tar |
| info | посмотреть информацию о системе |
| inspect | посмотреть информацию о контейнере |
| kill | остановить запущенный контейнер |
| load | загрузить образ из архива |
| login | авторизация в официальном репозитории Docker |
| logout | выйти из репозитория Docker |
| logs | посмотреть логи контейнера |
| pause | приостановить все процессы контейнера |

Продолжение таблицы 1 - Список подкоманд docker

|  |  |
| --- | --- |
| Команда | Описание |
| port | проброс портов для контейнера |
| ps | список запущенных контейнеров |
| putt | скачать образ контейнера из репозитория |
| push | отправить образ в репозиторий |
| rename | переименовать контейнер |
| restart | перезапустить контейнер |
| rm | удалить контейнер |
| rmi | удаляет один и более образов |
| run | выполнить команду в контейнере |
| save | сохранить образ в архив tar |
| search | поиск образов в репозитории по заданному шаблону |
| start | запустить контейнер |
| stats | статистика использования ресурсов контейнером |
| stop | остановить контейнер |
| tag | тегирование образа |
| top | посмотреть запущенные процессы в контейнере |
| unpause | проложить выполнение процессов в контейнере |
| update | обновляет конфигурацию одного или нескольких контейнеров |
| version | показывает информацию о версии Docker |
| wait | блокирует контейнер до его остановки, а затем выводит код завершения |

Ниже представлен разбор некоторых подкоманд Docker.

docker ps — показывает список запущенных контейнеров. Некоторые из полезных флагов:

-a / --all — список всех контейнеров (по умолчанию показывает только запущенные);

-q / --quiet — перечислить только id контейнеров (полезно, когда вам нужны все контейнеры).

docker pull — большинство образов создаётся на основе базового образа из Docker Hub. Docker Hub содержит множество готовых образов, которые можно использовать вместо того, чтобы создавать и настраивать свой собственный. Чтобы скачать определённый образ или набор образов (репозиторий), используйте команду docker pull.

docker build — эта команда собирает образ Docker из Dockerfile и «контекста». Контекст сборки — это набор файлов, расположенных по определённому пути или URL. Используйте флаг -t, чтобы задать имя образа. Например, команда docker build -t my\_container . соберёт образ, используя текущую директорию, на что указывает точка в конце.

docker run — запускает контейнер, на основе указанного образа. Эту команду можно дополнять другими, например, docker run my\_image -it bash запустит контейнер, а затем запустит в нём bash.

docker logs — эта команда используется для просмотра логов указанного контейнера. Можно использовать флаг --follow, чтобы следить за логами работающей программы: docker logs --follow my\_container.

docker volume ls — показывает список томов, которые являются предпочитаемым механизмом для сохранения данных, генерируемых и используемых контейнерами Docker.

docker rm — удаляет один и более контейнеров, например, docker rm my\_container.

docker rmi — удаляет один и более образов, например, docker rmi my\_image.

docker stop — останавливает один и более контейнеров. Команда docker stop my\_container остановит один контейнер, а docker stop $(docker ps -a -q) — все запущенные. Более грубый способ — использовать docker kill my\_container, который не пытается сначала аккуратно завершить процесс.

Можно комбинировать эти команды. Например, для очистки всех контейнеров и образов:

Останавливаем все запущенные контейнеры docker kill $(docker ps -q).

Удаляем все остановленные контейнеры docker rm $(docker ps -a -q).

Удаляем все образы docker rmi $(docker images -q).

Для просмотра всей информации о Docker используется следующая команда (рисунок 2.16).

**Команда: *docker info***

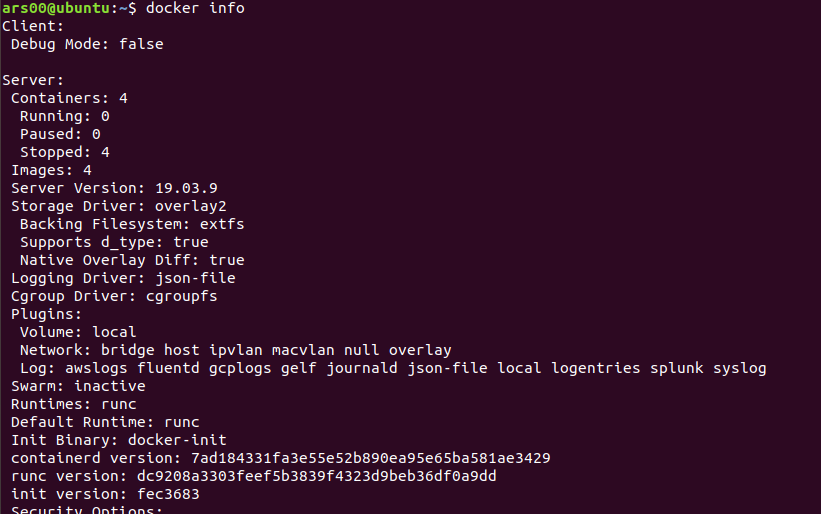


Рисунок 2.16 – Информация о docker

**2.7 Запуск контейнера Docker**

Контейнеры могут выполнять полезные действия, а также могут быть интерактивными. Контейнеры похожи на виртуальные машины, но являются менее требовательными к ресурсам.

Для поиска необходимых образов на Docker Hub (рисунок 2.17) используется команда docker и подкоманда search. Чтобы найти образ Ubuntu, нужно ввести:

**Команда: *docker search ubuntu***

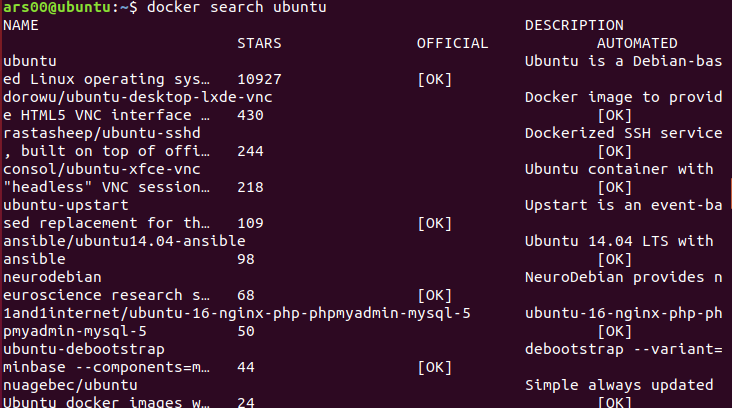


Рисунок 2.17 – Поиск в Docker Hub

Скрипт просматривает Docker Hub и возвращает список всех образов, имена которых подходят под заданный поиск.

Проще говоря, с помощью команды search был организован поиск, по ключевому слову Ubuntu, образов в Docker Hub.

В столбце OFFICIAL строка OK показывает, что образ построен и поддерживается компанией, которая занимается разработкой этого проекта.

После чего с помощью команды pull нужно выбрать необходимую вам версию и приступить к скачиванию (рисунок 2.18).

**Команда: *docker pull ubuntu***

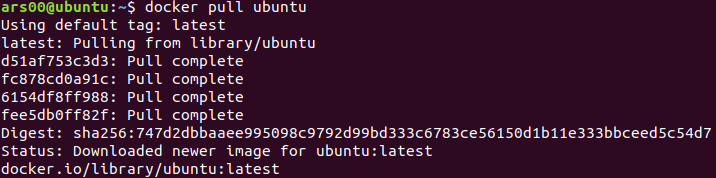


Рисунок 2.18 – Скачивание образа

Для просмотра загруженных на компьютер образов нужно ввести следующую команду (рисунок 2.19).

**Команда: *docker images***

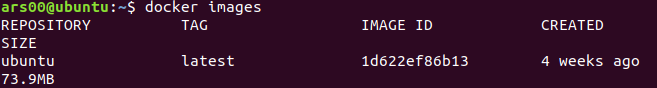


Рисунок 2.19 – список загруженных на компьютер образов

После загрузки образа можно запустить контейнер с загруженным образом с помощью подкоманды run (рисунок 2.20). Если при выполнении docker с помощью подкоманды run образ еще не загружен, клиент Docker сначала загрузит образ, а затем запустит контейнер с этим образом (рисунок 2.21).

В качестве примера запустим контейнер с помощью последней версии образа Ubuntu. Комбинация параметров -i и -t обеспечивает интерактивный доступ к командному процессору контейнера.

**Команда: *docker run -it ubuntu***



Рисунок 2.20 – Запуск контейнера через команду run

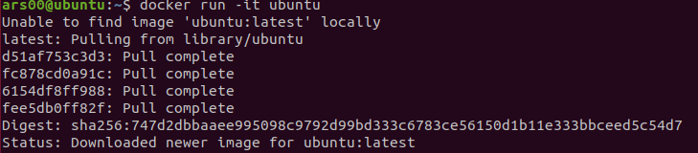


Рисунок 2.21 – Запуск контейнера и скачивание образа сразу через команду run

Далее обновите индекс пакетов с помощью уже известной вам команде (рисунок 2.22).

**Команда: *sudo apt-get update***

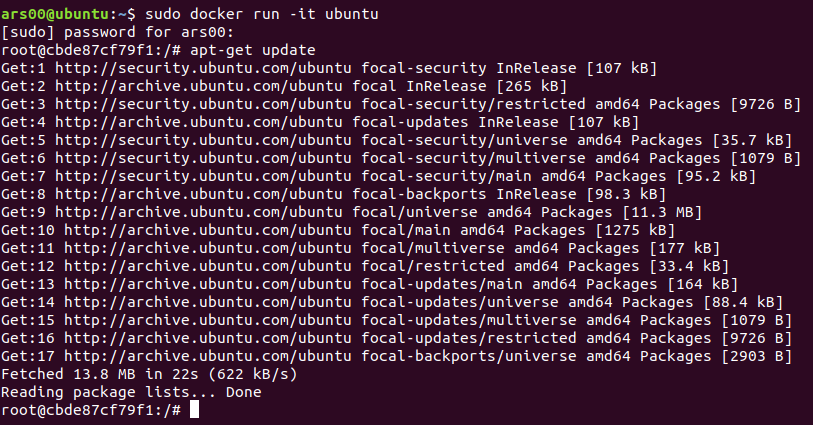


Рисунок 2.22 – Обновление индекса пакетов для гостевой ОС

**2.7.1 Установка и работа с Nano**

Для выполнения работы необходимо установить текстовый редактор Nano (рисунок 2.23).

Nano — текстовый редактор для консоли. Используется в Unix-подобных ОС, распространяется свободно.

**Команда: *apt install nano***

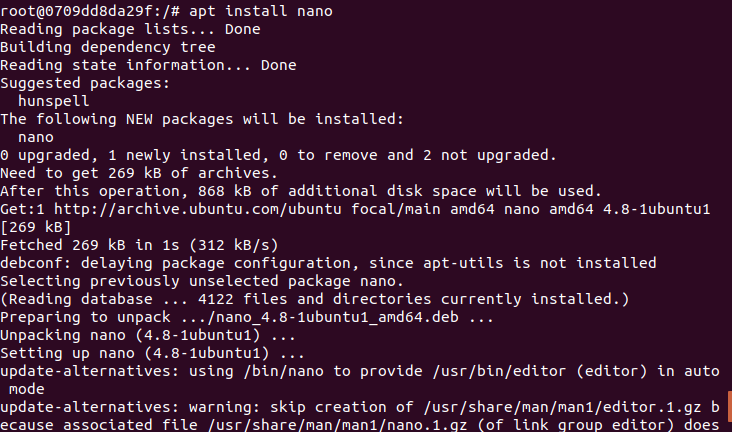


Рисунок 2.23 – Установка текстового редактора

Нужно заметить, что для разных ОС данный текстовый редактор устанавливается разными командами.

Например:

**CentOS, Fedora:**

**Команда: *yum install nano***

**Gentoo:**

**Команда: *install app-editors/nano***

**Debian, Ubuntu:**

**Команда: *apt-get install nano***

Данный текстовый редактор необходим для написания скриптов по вашему варианту.

Sh - это скрипт командной оболочки, созданный специально для операционной системы Linux/UNIX. Вы всегда можете запустить новый экземпляр оболочки bash, дав команду bash или sh. При этом можно заставить новый экземпляр оболочки выполнить какой-то скрипт, если передать имя скрипта в виде аргумента команды bash. Так, для выполнения скрипта myscript надо дать команду "sh myscript".

Для создания текстового файла используем следующую команду (рисунок 2.24).

**Команда: *nano «название файла».sh*** (в данном примере sp1)



Рисунок 2.24 – Создание файла

Проверим работу текстовых файлов с помощью команды echo (рисунок 2.25).

***ECHO “Hello World”***



Рисунок 2.25 – Вывод Hello World

Echo - это очень простая и в то же время часто используемая встроенная команда оболочки Bash. Она имеет только одно назначение - выводить строку текста в терминал, но применяется очень часто в различных скриптах, программах, и даже для редактирования конфигурационных файлов.

Для того чтоб сохранить изменения и выйти необходимо нажать сочетание клавиш Ctrl+X, затем Y.

Далее нужно проверить созданный нами файл с помощью команды ls (рисунок 2.26).



Рисунок 2.26 – Проверка созданного файла

Команда ls — вероятно наиболее используемая команда в любой UNIX-системе. Её предназначение — вывод информации о файлах и каталогах. Дополнительные опции команды позволяют получить более подробную информацию и сортировать её определённым образом.

Необходимо сделать данный файл исполняемым, для этого используем команду chmod (рисунок 2.27).

**Команда: *Chmod ugo+x “название файла”.sh***

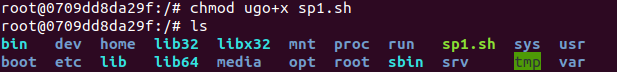


Рисунок 2.27 – Установка прав на файл

Система полномочий в Linux имеет очень важное значение, поскольку благодаря ей можно разделять привилегии между пользователями, ограничить доступ к нежелательным файлам или возможностям, контролировать доступные действия для сервисов и многое другое. В Linux существует всего три вида прав - право на чтение, запись и выполнение, а также три категории пользователей, к которым они могут применяться - владелец файла, группа файла и все остальные.

Эти права применяются для каждого файла, а поскольку все в системе Linux, даже устройства, считаются файлами, то, получается, что эти права применимы ко всему. Команде chmod, которая как раз используется для установки прав.

Эта команда имеет типичный для команд linux синтаксис, сначала команда, затем опции, а в конце файл или папка, к которой ее нужно применить:

$ chmod опции права /путь/к/файлу

Сначала рассмотрим какими бывают права доступа linux и как они устанавливаются. Есть три основных вида прав:

* r - чтение;
* w - запись;
* x - выполнение;
* s - выполнение от имени суперпользователя (дополнительный).

Также есть три категории пользователей, для которых вы можете установить эти права на файл linux:

* u - владелец файла;
* g - группа файла;
* о - все остальные пользователи.

Синтаксис настройки прав такой:

группа\_пользователейдействиевид\_прав

В качестве действий могут использоваться знаки "+" - включить или "-" - отключить. Рассмотрим несколько примеров:

* u+x - разрешить выполнение для владельца;
* ugo+x - разрешить выполнение для всех;
* ug+w - разрешить запись для владельца и группы;
* o-x - запретить выполнение для остальных пользователей;
* ugo+rwx - разрешить все для всех;

Проверка как работает пример скрипта (рисунок 2.28).

**Команда: *sh “название файла”.sh***



Рисунок 2.28 – Проверка работы скрипта

Чтобы выйти и остановить контейнер, введите команду exit (рисунок 2.29).

**Команда: *exit***



Рисунок 2.29 – Выход их контейнера

Чтобы посмотреть список всех контейнеров и их coin id (для подключения и запуска), нужно ввести следующую команду (рисунок 2.30).

**Команда: *docker ps -a***

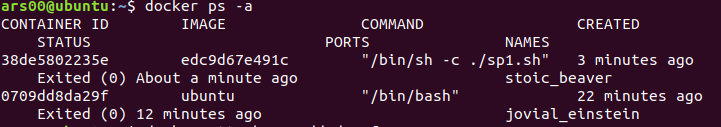


Рисунок 2.30 – Список всех контейнеров

Для запуска остановленного контейнера, нужно ввести следующую команду (рисунок 2.31):

**Команда: *sudo docker start “coin id”***



Рисунок 2.31 – Запуск остановленного контейнера

Подключение к существующему контейнеру осуществляется через команду (рисунок 2.32)

**Команда: *sudo docker attach “coin id”***



Рисунок 2.32 – Подключение к существующему контейнеру

**2.7.2 Создание и работа с Dockerfile**

Далее создадим Dockerfile с помощью текстового редактора nano (рисунок 2.33).

**Команда: *nano dockerfile***



Рисунок 2.33 – Создание Dockerfile.

Как говорилось выше, *Dockerfile* – это подробный пошаговый сценарий для автоматизации сборки. В файлах Dockerfile содержатся инструкции по созданию образа. С них, набранных заглавными буквами, начинаются строки этого файла. После инструкций идут их аргументы. Инструкции, при сборке образа, обрабатываются сверху вниз.

Ниже приведен список инструкций Dockerfile с краткими комментариями:

* FROM — задаёт 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 лишь 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.

После создания dockerfile откроется окно где мы можем приступать к написанию кода (рисунок 2.34).

***FROM ununtu***

***COPY sp1.sh .***

***RUN cgmod ugo+x sp1.sh***

***CMD ./sp1.sh***



Рисунок 2.34 – Dockerfile

Файл Dockerfile должен начинаться с инструкции FROM, или с инструкции ARG, за которой идёт инструкция FROM. Ключевое слово FROM сообщает Docker о том, чтобы при сборке образа использовался бы базовый образ, в нашем случае Ubuntu.

Инструкция COPY представлена в нашем файле так: COPY sp1.sh. Она сообщает Docker о том, что нужно взять информацию из файла sp1 и добавить их в текущую рабочую директорию образа.

Инструкция RUN позволяет создать слой во время сборки образа. После её выполнения в образ добавляется новый слой, его состояние фиксируется. Инструкция RUN часто используется для установки в образы дополнительных пакетов. В примере выше этим «слоем» является установка прав.

Инструкция CMD предоставляет Docker команду, которую нужно выполнить при запуске контейнера.

А точка в конце (sp1.sh **.**) означает что Докерфайл находится в той же категории, из которой выполняется команда.

После чего стоит собрать образ (рисунок 2.35).

**Команда: *doker build -t test .***

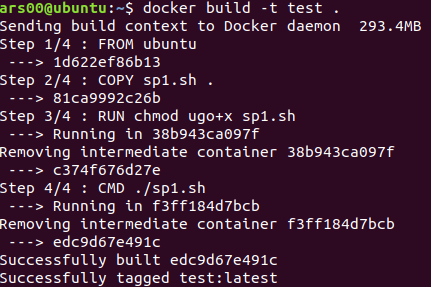


Рисунок 2.35 – Сборка образа

Docker build — эта команда собирает образ Docker из Dockerfile и «контекста». Контекст сборки — это набор файлов, расположенных по определённому пути или URL. Используйте флаг -t, чтобы задать имя образа. Например, команда docker build -t my\_container. соберёт образ, используя текущую директорию, на что указывает точка в конце.

**Команда: *docker run -it***



Рисунок 2.36 – Запуск контейнера

Docker run — запускает контейнер, на основе указанного образа. Прошу обратить внимание откуда был взят последняя часть скрипта (предыдущий скрин). после чего сработал ранее прописанный скрипт.

**2.8 Установка Docker для Windows**

Docker Desktop для Windows — это нативное Windows-приложение, которое предоставляет среду для сборки и запуска контейнеризированных приложений. В его основе используется нативная Hyper-V виртуализация и сети, что является наиболее быстрым и стабильным вариантом для работы с Docker в Windows. Docker Desktop для Windows поддерживает как Windows-, так и Linux-контейнеры.

Нам потребуется операционная система Windows 10 версии Корпоративная, Профессиональная или Образовательная. Также должны быть подключены компоненты Hyper-V и Windows Containers. Для работы с этими компонентами система должна выполнять следующие требования:

* 64-разрядная операционная система;
* не менее 4 ГБ оперативной памяти;
* поддержка аппаратной виртуализации на уровне BIOS.

Проверить, включена ли у вас виртуализация можно одним из ниже приведенных способов.

Через диспетчер задач, во вкладке «производительность» (рисунок 2.37).

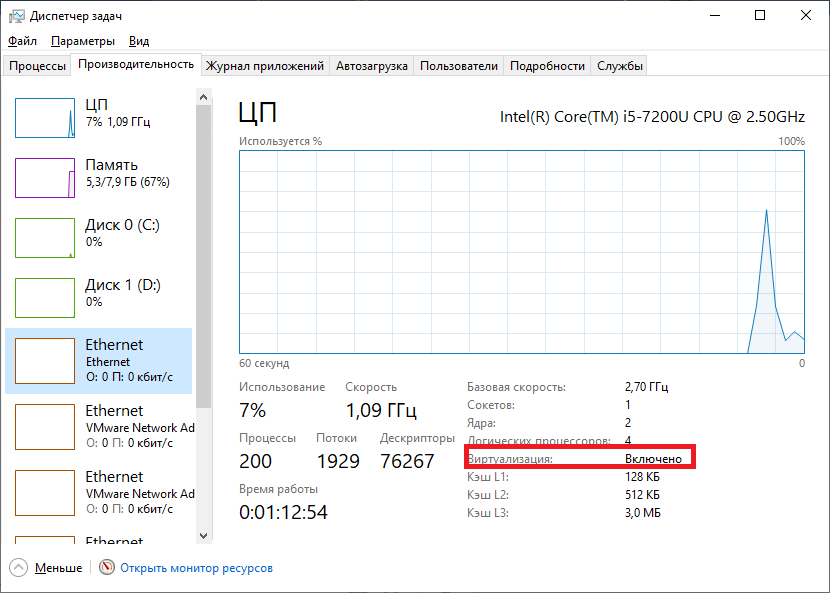


Рисунок 2.37 – Проверка включения виртуализации через диспетчер задач

Для проверки по второму способу необходимо запустить командную строку (рисунок 2.38). Сочетание клавиш Win + R, после чего выполняем «cmd» (рисунок 2.39).

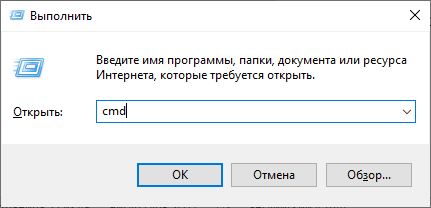


Рисунок 2.38 – Запуск командной строки

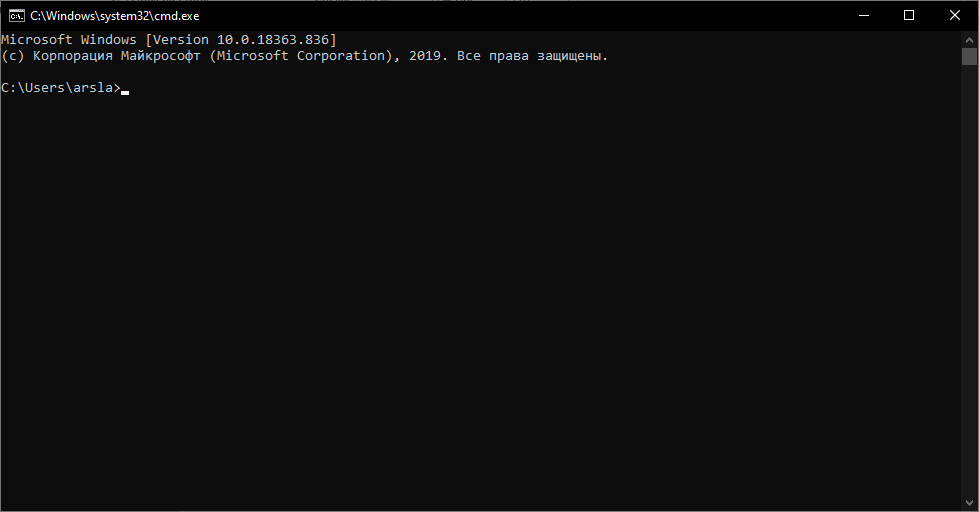


Рисунок 2.39 – Командная строка

Далее прописываем команду необходимую для проверки виртуализации (рисунок 2.40).

**Команда:** ***systeminfo***



Рисунок 2.40 – Команда systeminfo

Нас интересует последний пункт — «Требования Hyper-V», где всеми значениями должно быть «Да» (рисунок 2.41).

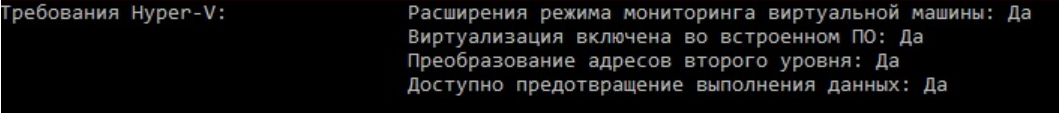


Рисунок 2.41 – Проверка виртуализации через командную строку

Существует два варианта установки Docker для Windows:

* Приложение «Docker Toolbox for Windows»;
* приложение «Docker for Windows».

Docker Toolbox для Windows является устаревшим решением, которым вы можете воспользоваться, если ваш компьютер не удовлетворяет требования, которые нужны для Docker для Windows.

В этом уроке мы будем устанавливать Docker через приложение «Docker for Windows», так как это решение является наиболее стабильным и надёжным. Итак, сначала нужно скачать установщик. Скачать его можете по этим ссылкам:

* https://docs.docker.com/docker-for-windows/install/
* https://ravesli.com/ustanovka-docker-v-windows/

Далее запустите инсталлятор Docker Desktop Installer.exe. Следуйте инструкциям мастера установки, дайте установщику разрешение на использование прав администратора и дождитесь окончания установки.

Для Docker права администратора нужны для того, чтобы установить сетевые компоненты, создать ссылки на приложения Docker и управлять Hyper-V виртуальными машинами.

Запустите Docker Desktop и начните его установку (рисунок 2.42).

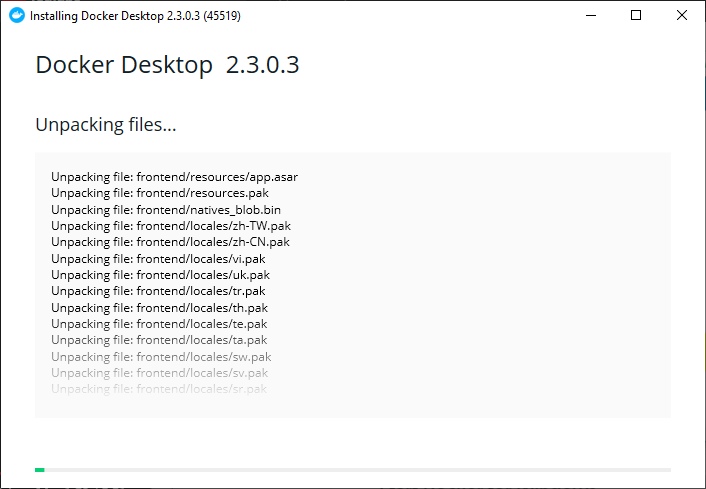


Рисунок 2.42 – Процесс установки Docker Desktop

По завершению установки необходимо будет перезапустить компьютер, после чего можно приступать к работе с Docker Desktop (рисунок 2.43).

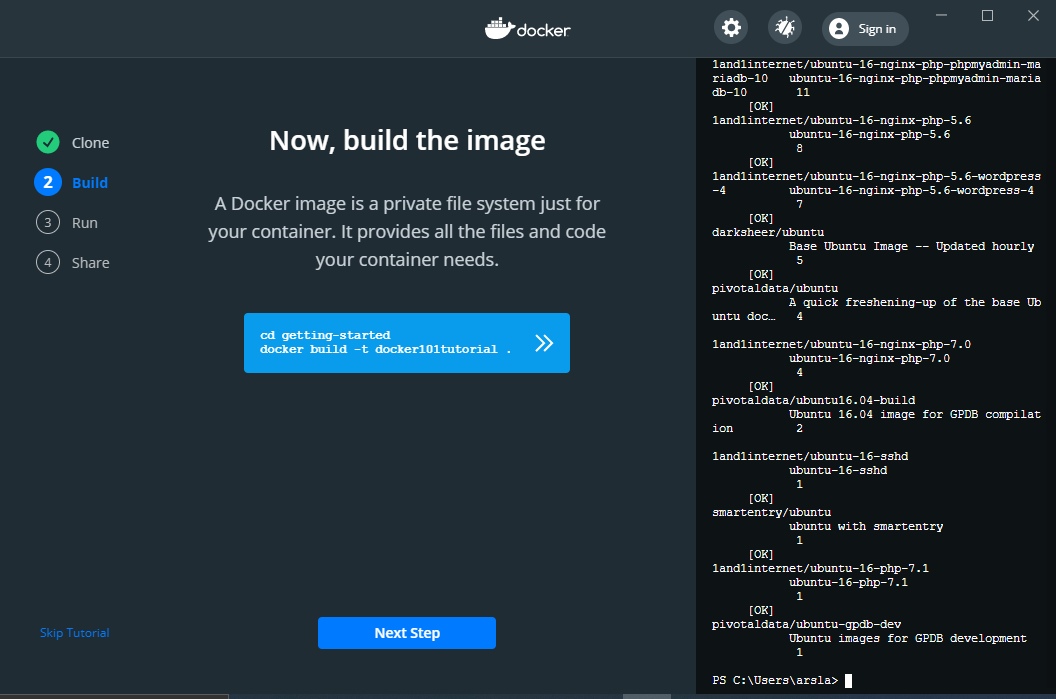


Рисунок 2.43 – Docker Desktop

# 3 Варианты заданий

Варианты заданий (образ ОС для Docker)

1. CentOS
2. Ubuntu
3. Debian
4. Alpine
5. Amazon Linux

Все задания необходимо выполнить по следующим правилам:

* + Скрипт должен при запуске выводить: фамилию и имя автора, название программы, краткое описание
  + Скрипт должен выводить подсказки для пользователя, помогающие пользователю корректно работать с программой
  + Скрипт должен работать в режиме диалога с пользователем (в виде “вопрос”-“ответ”)
  + Скрипт должен обрабатывать аварийные ситуации, ошибки и некорректно введённые данные
    - сообщения об ошибках должны выводиться в поток stderr
  + Скрипт должен быть организован в виде бесконечного цикла
    - после выполнения операций скрипт должен предлагать пользователю начать выполнение сначала или выйти из цикла
    - если на последней итерации выводилось сообщение об ошибке и пользователь закончил работу скрипта, скрипт должен

завершиться с кодом возврата

Пример диалога пользователя с программой:

$ ./program.sh

Программа поиска пользователей

С помощью данной программы вы можете искать пользователя по его UID.

Разработчик: Иванов Иван

Введите UID пользователя: 100000

Ошибка! Пользователь с таким UID не найден. Хотите продолжить? (y/n)

y

Введите UID пользователя:

-1

Ошибка! UID должен быть неотрицательным числом. Хотите продолжить? (y/n)

y

Введите UID пользователя: 1000

Пользователь: user Хотите продолжить? (y/n)

n

......Выход из программы….

Варианты скриптов:

Задание 1

Разработать скрипт, который:

• запрашивает у пользователя имя пакета

• если указанный пакет установлен, то выводит информацию о пакете

• если указанный пакет не установлен, то выводит сообщение об этом и производит поиск пакета в репозиториях

• если пакет найден в репозиториях, то предлагает пользователю его установить или выйти из скрипта

Для выполнения задания используйте команды rpm и yum.

# 4 Справочник по командам BASH

***RPM***

Rpm (RPM Package Manager) – это менеджер RPM-пакетов в Red Hat подобных системах. Позволяет выполнять установку, удаление и обновление программного обеспечения. Среди основных преимуществ этого менеджера следует отметить:

* Простоту процесса обновления, удаления пакетов;
* возможность проверки GPG подписи и контрольной суммы для проверки целостности пакета.

Ниже представлена таблица (таблица 2) с базовыми командами, которые обычно требуются в работе с RPM менеджером.

Таблица 2 – Базовые команды, которые требуются в работе с RPM

|  |  |
| --- | --- |
| Команда | Описание |
| rpm --import RPM-GPG-KEY | Импорт GPG ключа. Необходимо для проверки пакетов. |
| rpm -qip packagename.rpm | Просмотр информации о пакете packagename.rpm, который расположен в текущей директории. |
| rpm -qp --scripts packagename.rpm | Просмотр скриптов %pre- %post- install (uninstall). |
| rpm -qp --changelog packagename.rpm | Просмотр changelog пакета |
| rpm -qR packagename | Просмотр зависимостей необходимых для указанного пакета. |
| rpm -ivh packagename.rpm | Установка пакета. |
| rpm -qi packagename | Просмотр информации о пакете, который уже установлен в системе. |
| rpm -Uvh packagename.rpm | Обновление или установка пакета. |

Продолжение таблицы 2 - Базовые команды, которые требуются в работе с RPM

|  |  |
| --- | --- |
| rpm -Fvh packagename.rpm | Обновление пакета. Для случая, если предыдущая его версия уже установлена в системе. |
| rpm -e packagename | Удаление пакета. |
| rpm -qa --last | Просмотр даты инсталляции пакетов. |
| rpm -qa | Просмотр списка всех пакетов, установленных в системе. |
| rpm -qa «client\*» | Поиск всех пакетов, начинающихся на client. |
| rpm -qf /etc/rc.conf | Распознавание принадлежности файла rc.conf какому-либо пакету. |

***Yum***

Yum (Yellow dog Updater, Modified) является открытым пакетным менеджером в Red Hat подобных системах. Был создан с целью облегчения процесса обновления системы с учетом взаимосвязей пакетов. Также, Yum осуществляет поиск RPM-пакетов в репозиториях, их установку, отслеживание зависимостей между пакетами, удаление не использующихся пакетов, а также даунгрейд (откат версии пакета до предыдущей).

По умолчанию управление менеджером выполняется через консоль, однако есть возможность установить дополнительные компоненты для работы через графический интерфейс (PackageKit).

Общий синтаксис команды:

yum [параметр] [команда] [имя\_пакета]

Таблица 3 – Базовые командыyum

|  |  |
| --- | --- |
| Команда | Описание |
| yum install packagename | Для установки на VPS хостинг в систему какого-либо пакета |
| yum install package1 package2 … | Установка сразу нескольких пакетов, указанных через пробел |
| yum install package.rpm | Установка пакета из файла .rpm |
| yum remove packagename | Удаление пакета |
| yum list packagename | Поиск пакета в репозиториях по названию |
| yum search vpn | Поиск пакета в репозиториях по названию и содержанию (пример: vpn) |
| yum provides client | Поиск пакета в репозиториях по названию содержимого файла (пример: client) |
| yum update | Обновление системы для всей системы |
| yum update packagename | Обновление системы для определенного пакета |
| yumdownloader packagename | Скачивание пакета без установки |
| yum deplist packagename | Вывод зависимостей пакета |

**5 Контрольные вопросы**

1. Плюсы и минусы современных ОС основанных на UNIX.
2. Что такое Docker, Dockerfile и контейнер?
3. Чем Docker отличается от простых виртуальных машин (VMware, Oracle)?
4. Что из себя представляет BASH?
5. Какие команды BASH вы знаете?
6. Какие подкоманды Docker вы использовали?
7. Что из себя представляет команда Curl?
8. Почему на Ubuntu писали sudo?
9. Какую функцию несет «**.**» в скрипте?

**6 Список литературы**

1.Что такое Docker и как его использовать в разработке [Электронный курс] – Режим доступа: <https://eternalhost.net/blog/razrabotka/chto-takoe-docker>

2.Открытое программное обеспечение [Электронный курс] – Режим доступа: <https://losst.ru/?s=stat>

3.Изучаем Docker [Электронный курс] – Режим доступа: <https://habr.com/ru/company/ruvds/blog/438796/>

4.Установка Docker [Электронный курс] – Режим доступа: <https://www.hostinger.ru/rukovodstva/ustanovka-docker-v-ubuntu-18-4/>

5.Bash скрипты [Электронный курс] – Режим доступа: <https://habr.com/ru/company/ruvds/blog/325522/>

6.ОС ТУСУР 02 2013 [Электронный ресурс] - Режим доступа: <https://storage.tusur.ru/files/40669/rules_gum_02-2013.pdf>