МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Программные средства контейнеризации образов Unix – систем на платформе «Docker»

Студент	15.06.20 (подпись, дата)	Денисов Артем Олегович (ФИО полностью)
Руководитель	15.06.20 (подпись, дата)	Балашова И. Ю. (фамилия, инициалы)
Нормоконтролёр	_____ (подпись, дата)	Такташкин Д. В. (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ (подпись) Макарычев П. П. (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ (подпись) Попова Н. А. (фамилия, инициалы)

Пенза, 2020

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет Вычислительной техники

Кафедра МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____П.П. Макарычев
«__»_____20__ г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Денисов Артем Олегович гр. 16ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>
2.Тема работы Программные средства контейнеризации образов Unix – систем на платформе «Docker»

Тема утверждена приказом ПГУ № 1488/о от "30" декабря 2019

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Балашова И. Ю.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение: программное средство обеспечивает конвертирование образов виртуальных в контейнеры на платформе «Docker»

Основные функции:
1) Проверка образа виртуальной системы на корректность
2) Конвертирование образов виртуальных машин
3) Формирование Dockerfile машины
4) Формирование YAML файла машины
5) Автоматический запуск контейнера с необходимыми параметрами

Языки программирования: Bash, Python
Инструментальные средства разработки: Visual Studio Code, JetBrains PyCharm, WSL v. 2
Среда исполнения: Linux, Docker

5. Перечень подлежащих разработке вопросов
1) Анализ предметной области и постановка задачи
2) План разработки и оценка бюджета
3) Проектирование программных средств
4) Реализация программных средств
5) Тестирование и анализ качества программного кода
6) Рефакторинг программных средств
7) Оформление пояснительный записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	5	13.04.2020	
2	План разработки и оценка бюджета	5	13.04.2020	
3	Проектирование программных средств	25	01.05.2020	
4	Реализация программных средств	35	20.05.2020	
5	Тестирование и анализ качества программного кода	10	02.06.2020	
6	Рефакторинг программных средств	5	10.06.2020	
7	Оформление пояснительный записки	15	13.06.2020	

Дата выдачи задания 30.12.2019

Руководитель бакалаврской работы _____ 30.12.2019
(подпись, дата)

Задание к исполнению принял студент _____ 30.12.2019
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 65 листов, 29 рисунков, 2 таблицы, 15 использованных источников, 2 приложения.

DOCKER, КОНТЕЙНЕРИЗАЦИЯ, ВИРТУАЛИЗАЦИЯ, BASH-СКРИПТИНГ, PYTHON, LINUX, VIRTUALBOX, VMWARE, UBUNTU, DOCKERFILE, YAML.

Объект исследования – автоматическая Docker контейнеризация.

Цель выпускной квалификационной работы – разработка программных средств для конвертирования образов виртуальных машин и использования их в среде Docker.

Среда исполнения – Linux, Docker.

Разработка программных средств проводилась на языках программирования Bash и Python в средах Visual Studio Code и JetBrains PyCharm в операционной системе Windows. Тестирование и отладка проводилась на основе Docker в операционной системе Ubuntu Server 18.04 LTS семейства Linux.

Содержание

Введение ... 6
1 Разработка программных средств для реализации автоматического конвертирования образов систем ... 7
1.1 Анализ предметной области ... 7
1.2 Сравнительный анализ программных продуктов-аналогов ... 8
1.2.1 Анализ Image2Docker ... 9
1.2.2 Анализ RancherVM ... 9
1.2.3 Результаты проведенного анализа ... 11
1.3 Постановка задачи ... 11
1.4 Анализ требований программного средства автоматического конвертирования образов систем ... 12
1.4.1 Функциональные требования программного средства ... 13
1.4.2 Нефункциональные требования программного средства ... 17
2 Проектирование программных средств ... 18
2.1 Проектирование виртуализации дискового пространства исходного образа системы ... 18
2.2 Проектирование проверки требований используемой операционной системы ... 22
2.3 Проектирование генерации Dockerfile для управления платформой Docker ... 23
3 Реализация программного средства для автоматического конвертирования образов систем ... 27
4 Тестирование программного средства ... 31
4.1 Рефакторинг кода ... 33
5 Планирование разработки и оценка бюджета ... 35
Заключение ... 39
Список использованных источников ... 41
Приложение А Глоссарий ... 43
Приложение Б Листинг программы ... 45

Введение

«Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Позволяет «упаковать» приложение или группу приложений со всем окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему, а также предоставляет среду по управлению контейнерами» [1].

Контейнер – это изолированная от основной операционной системы совокупность процессов. Основным и главным отличием от виртуальных машин является виртуализация процессов, а не всей системы целиком. Тем самым, напрямую используются ресурсы основной машины, как подпроцесс без какой-либо эмуляции «машины в машине».

Данная работа является актуальной, так как существующие программные средства не обеспечивают удобства и полной универсальности в переносе связанных многомодульных систем. На смену виртуальных машин приходят контейнеры, которые осуществляют нагрузку на основную операционную систему намного меньшую, чем их конкурент. Данная технология является одним из приоритетов разработки крупных фирм, таких как Google и Amazon, начиная с 2014 года.

Ключевыми особенностями использования Docker являются:

– экономия значительных ресурсов ЭВМ при контейнеризации;

– перенос готовых контейнеров на другие машины осуществляется без использования сторонних модулей, то есть, если меняется операционная система хост-машины, контейнер будет работать после его переноса;

– виртуализация не всей операционной системы целиком, а лишь нужных процессов, которые необходимо добавить в контейнер.

Целью выпускной квалификационной работы является разработка программных средств для конвертирования образов виртуальных машин и использования их в среде Docker.

1 Разработка программных средств для реализации автоматического конвертирования образов систем

1.1 Анализ предметной области

Контейнеры похожи на директории. В контейнерах содержится все, что нужно для работы приложения. Каждый контейнер создается из образа. Контейнеры могут быть созданы, запущены, остановлены, перенесены или удалены. Каждый контейнер изолирован и является безопасной платформой для приложения. Контейнеры — это компонента работы [2].

Самыми популярными категориями для контейнеризации программ являются [3]:

– базы данных;
– операционные системы;
– языки программирования;
– программы инфраструктуры.

К первой категории из популярных баз данных, использующихся в контейнерах, можно отнести:

– Oracle Database Enterprise Edition;
– MySQL Server Enterprise Edition;
– PostgreSQL;
– Redis;
– MongoDB.

Из состава операционных систем, которые уже перенесены в контейнеры, можно отнести:

– Alpine;
– Ubuntu;
– CentOS;
– Debian;
– OpenSUSE.

Под языками программирования в контейнерах понимается установленная операционная система (ОС) Ubuntu с единственно установленным пакетом программного средства и необходимыми зависимостями для его работы. В общем доступе представлены следующие популярные контейнеры:

– Python;
– GoLang;
– PHP;
– Ruby;
– Java;
– Perl;
– Bash;
– Erlang.

К разделу программ инфраструктуры можно отнести следующие программные средства:

– Nginx;
– Apache HTTP Server;
– Node.js;
– Nextcloud;
– Apache TOMEE;
– Tomcat.

Подводя итог уже готовых контейнеров, можно увидеть, что переход с виртуальных машин на контейнеры системы Docker можно произвести вручную без каких-либо проблем с переносом данных.

1.2 Сравнительный анализ программных продуктов-аналогов

В соответствии с полученной классификацией популярных готовых контейнеров, было решено изучить программы для автоматического переноса образов виртуальных машин в среду Docker. На данный момент на рынке

имеется несколько программ для переноса данных из образа виртуальной машины в контейнер Docker.

1.2.1 Анализ Image2Docker

Image2Docker является самым популярным продуктом по решению вопроса конвертирования образов. Он был написан сотрудником Docker Inc. с исходным кодом в общем доступе на сайте GitHub [4, 5].

Данное программное средство обладает рядом преимуществ и недостатков.

Преимущества:

– переносит только конфигурационные файлы и базы данных, если таковые имеются;

– имеется поддержка Windows Server;

– возможность создания Hyper-V контейнеров, которая положительно сказывается на производительности работы в среде Docker.

Недостатки:

– не работает с операционными системами Linux семейства;

– не поддерживается разработчиком с 2017 года, имеются проблемы с новыми версиями Docker;

– нет возможности переноса необходимых под конкретную задачу конфигурационных данных и нет возможности доработки отдельного модуля переноса.

Таким образом, можно сделать вывод, что данное программное средство является узконаправленным решением перехода к контейнеризации своих виртуальных машин и подойдет только для обладателей виртуальных машин на операционной системе Windows.

1.2.2 Анализ RancherVM

Еще одним программным средством является RancherVM, разработанная коллективом Rancher в 2018 году [6, 7].

Данное программное средство, в отличие от конкурента, реализует автоматическую контейнеризацию покластерно, то есть каждой необходимой программе выделяется свой независимый контейнер. Все контейнеры автоматически подключаются в одну подсеть для взаимодействия между собой.

Данное программное средство обладает рядом преимуществ и недостатков.

Преимущества:

– имеет поддержку автоматической кластеризации;

– нативная поддержка Linux;

– имеет совместимость с Kernel Virtual Machine, что исключает проблемы с работоспособностью перенесенного в контейнер ПО;

– имеет встроенный GUI для удобного управления контейнерами.

Недостатки:

– данная разработка функционирует только при использовании дополнительных программ для кластеризации контейнеров, например, таких как Kubernetes;

– нет поддержки операционной системы Windows;

– проект является замороженным с конца 2019 года;

– многие ошибки в приложении так и не были исправлены;

– полное копирование программы с виртуальной машины может привести к иррациональному использования дискового пространства и ресурсов процессора.

Таким образом, можно сделать вывод, что данное программное средство является более универсальным и практичным, чем Image2Docker, но имеет свои недостатки. Главным вопросом остается замороженность проекта, так как ошибок для данной версии продукта осталось слишком много. Примером может служить проблема, которая так и не была исправлена с 2018 года, которая мешает переносу с некоторых виртуальных машин [8].

1.2.3 Результаты проведенного анализа

В результате проведенного анализа и имеющихся программных решений, было принято решение разработать средство для автоматического переноса виртуальных машин в Docker контейнеры.

Существующие программные средства не обеспечивают удобства и полной универсальности в переносе. Постоянно нужно выбирать между двумя продуктами, а также не исключено, что программное средство завершит перенос с ошибкой. Особенный риск имеют новые системы, так как разработка программного средства RancherVM остановилась в 2018 году, что означает непроверенную совместимость с новыми дистрибутивами, например Ubuntu 20.04 LTS или Linux Mint 19.

В связи с вышесказанным было принято решение разработать программное средство, независимое от дистрибутива Linux. Для реализации данного подхода были использованы скриптовые языки Bash и Python, которые являются предустановленными во все операционные системы подсемейства Linux.

1.3 Постановка задачи

В результате анализа предметной области было принято решение разработать программное средство для автоматического конвертирования образов систем в Docker контейнеры, которое может использоваться на любом дистрибутиве Linux подсемейства DEB.

Для решения данной задачи необходимо автоматическое управление образом преобразуемой системы и платформой Docker. Было решено использовать два популярных формата виртуальных машин:

– VDI (Virtual Disk Image), использующийся в Oracle VirtualBox;

– VMDK (Virtual Machine Disk), использующийся в VMware Workstation.

В рамках данной работы будут рассмотрены и проанализированы следующие этапы построения программного средства:

– виртуализация дискового пространства исходного образа системы;

– проверка требований используемой операционной системы;

– генерация Dockerfile для управления платформой Docker.

1.4 Анализ требований программного средства автоматического конвертирования образов систем

Анализ требований – один из важнейших этапов разработки программного обеспечения, включающий в себя сбор и систематизация требований к будущей системе [9].

Согласно поставленной задаче, необходимо перенести программные средства, установленные в образе виртуальной машины, в Docker контейнеры.

Для переноса одной программы в один контейнер не потребуется большого количества времени. Но если виртуальная машина являлась комплексным решением многих задач, на которой была установлена связка разных программ, переход на контейнеры может быть проблематичным и затянутым. Именно в данном случае и возникает идея оптимизации данного процесса.

Есть несколько путей решения данной проблемы:

– полный перенос данных с виртуальной машины в контейнеры с сохранением версии операционной;

– перенос конфигурационных файлов приложений с сохранением версий программы.

Самым простым и достаточно затратным является первый вариант с полным копированием системных данных. Данное решение подойдет для единоразового переноса данных виртуальных машин с некритичным статусом, так как вместе с необходимыми данными, в любом случае, перенесутся ненужные данные. Данный подход может захламить машину ненужным мусором, тем самым можно потерять быстродействие системы, даваемое контейнером. Таким образом, можно понять, что это не является

универсальным подходом к автоматизации переноса данных с машины в контейнеры.

Если же переносить только конфигурационные данные в контейнер, то экономится время переноса и объем данных, но при этом возрастает сложность алгоритма написания программы для переноса данных, так как необходимо узнать не только операционную систему данной виртуальной машины, но и все необходимые установленные модули, их версии и пути к необходимым конфигурационным файлам. Так же данное решение не может быть универсальным, так как для каждой программы, из которой необходимо перенести конфигурационные файлы, необходимо написать отдельный модуль.

Можно сделать вывод, что оптимальным вариантом среди двух будет являться перенос только конфигурационных файлов. Данный подход является тяжелым в реализации, но зато он будет прост в дальнейшей поддержке продукта, так как для новой программы необходимо будет написать отдельный компонент и не переписывать сам исходный код. Именно данный подход и будет реализован в ходе выполнения квалификационной работы.

1.4.1 Функциональные требования программного средства

Согласно поставленным задачам, были выделены следующие функциональные требования:

- проверить образ виртуальной системы;
- проверить системные требования ОС;
- выполнить конвертирование образа;
- сгенерировать Dockerfile по образу виртуальной машины;
- сгенерировать YAML файл по образу виртуальной машины.

На рисунке 1 представлена диаграмма вариантов использования. На диаграмме представлены все функциональные требования в виде вариантов использования и возможных взаимодействий внутри программы. Данные требования были выделены при анализе требований программного средства.

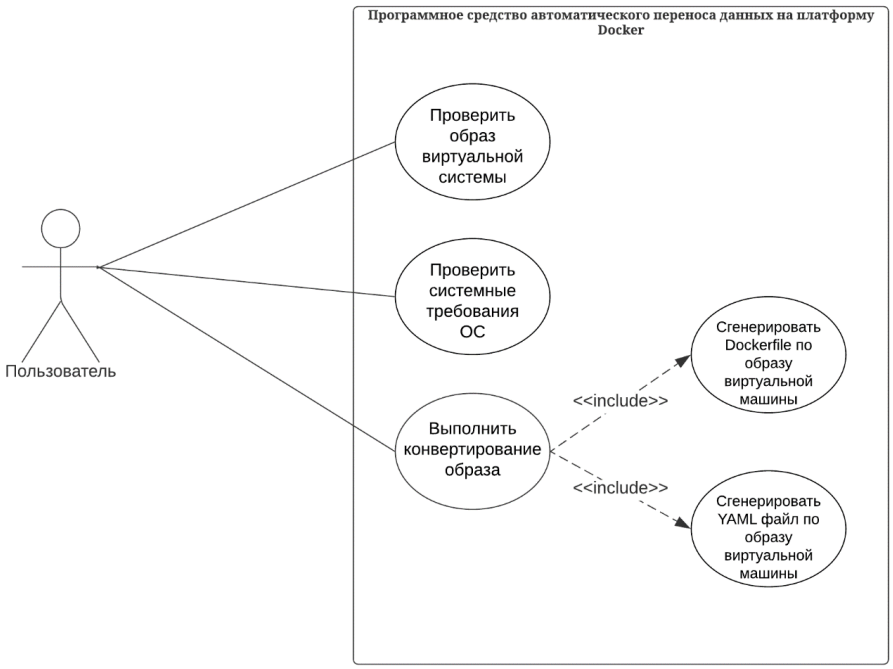


Рисунок 1 – Диаграмма вариантов использования

Спецификации прецедентов позволяют лучше понять последовательность действий, выполняемых актером при использовании конкретных функций системы. Спецификация прецедента «Проверить образ виртуальной системы» представлена на рисунке 2.

Прецедент: Проверить образ виртуальной системы
ID: 1
Краткое описание: Пользователь проверяет образ виртуальной системы на допустимость.
Главные актёры: Пользователь.
Второстепенные актёры: Нет
Предусловия: Нет
Основной поток: 1. Пользователь вводит путь до образа виртуальной системы 2. Система возвращает результат выполнения
Постусловия: Результат проверки образа передается пользователю
Альтернативные потоки: Нет.

Рисунок 2 – Спецификация прецедента «Проверить образ виртуальной системы»

На рисунке 3 представлена спецификация прецедента «Проверить системные требования ОС».

Прецедент: Проверить системные требования ОС
ID: 2
Краткое описание: Пользователь проверяет системные требования своей операционной системы
Главные актёры: Пользователь
Второстепенные актёры: Нет
Предусловия: Нет
Основной поток: 1. Пользователь вводит ключ для проверки своей ОС 2. Система возвращает результат выполнения
Постусловия: Результат проверки операционной системы передается пользователю
Альтернативные потоки: Нет

Рисунок 3 – Спецификация прецедента «Проверить системные требования ОС»

На рисунке 4 представлена спецификация прецедента «Выполнить конвертирование образа».

Прецедент: Выполнить конвертирование образа
ID: 3
Краткое описание: Пользователь выполняет преобразование виртуальной машины в Docker контейнер
Главные актёры: Пользователь
Второстепенные актёры: Нет
Предусловия: Нет
Основной поток: 1. Пользователь выбирает образ виртуальной системы из проверенных 2. Пользователь выбирает вариант компоновки виртуальной машины в контейнере 3. include «Сгенерировать Dockerfile по образу виртуальной машины» 4. include «Сгенерировать YAML файл по образу виртуальной машины» 5. Пользователю в виде YAML отчета выдается результат проделанной работы
Постусловия: Запускается Docker контейнер
Альтернативные потоки: Нет

Рисунок 4 – Спецификация прецедента «Выполнить конвертирование образа»

На рисунке 5 представлена спецификация прецедента «Сгенерировать Dockerfile по образу виртуальной машины».

Прецедент: Сгенерировать Dockerfile по образу виртуальной машины
ID: 4
Краткое описание: Программа генерирует Dockerfile для создания контейнера
Главные актёры: Пользователь
Второстепенные актёры: Нет
Предусловия: Нет
Основной поток: 1. Пользователь передает исходный образ с необходимой компоновкой 2. Система по заданным критериям собирает Dockerfile 3. Система возвращает Dockerfile и запускает его для создания контейнера
Постусловия: Dockerfile передан для просмотра пользователю
Альтернативные потоки: Нет

Рисунок 5 – Спецификация прецедента «Сгенерировать Dockerfile по образу виртуальной машины»

На рисунке 6 представлена спецификация прецедента «Сгенерировать YAML файл по образу виртуальной машины».

Прецедент: Сгенерировать YAML файл по образу виртуальной машины
ID: 5
Краткое описание: Программа генерирует YAML файл для отображения пользователю работы программы
Главные актёры: Пользователь
Второстепенные актёры: Нет
Предусловия: Нет
Основной поток: 1. Программа получает действия, производимые в Dockerfile 2. Программа генерирует YAML со всеми изменениями, произошедшими при конвертации 3. Система возвращает результат
Постусловия: Пользователь получил результат
Альтернативные потоки: Нет

Рисунок 6 – Спецификация прецедента «Сгенерировать YAML файл по образу виртуальной машины»

В результате анализа функциональных требований были выделены варианты использования программы. Для основных вариантов использования была приведена спецификация, которая позволяет понять последовательность действий, которые выполняет пользователь при использовании функций системы.

1.4.2 Нефункциональные требования программного средства

Согласно поставленным задачам, данное программное средство должно быть универсальным для всего семейства дистрибутивов DEB. Исходя из этого требования, для разработки были выбраны скриптовые языки программирования Bash и Python, которые используются и являются неотъемлемой частью дистрибутивов Linux/DEB. Также, платформа Docker, в большинстве своем, используется операционными системами семейства GNU.

2 Проектирование программных средств

2.1 Проектирование виртуализации дискового пространства исходного образа системы

Одним из важных компонентов программы является оценка загружаемого файла для дальнейшего переноса программ в Docker контейнеры. Для точного определения образа системы используется библиотека libguestfs-tools.

libguestfs – это набор инструментов для доступа и изменения образов дисков виртуальных машин. Его можно использовать для просмотра и редактирования файлов внутри гостевых машин, написания сценариев изменений для виртуальных машин, мониторинга использования диска, создания гостевых машин типа P2V и V2V, выполнения резервного копирования, клонирования виртуальных машин, создания виртуальных машин, форматирования дисков, изменения размера дисков и многого другого.

libguestfs может получить доступ практически к любому образу диска. Он может сделать это безопасно – без необходимости root прав и с несколькими уровнями защиты от образов мошеннических дисков. Он может получить доступ к образам дисков на удаленных компьютерах или на USB-накопителях, к проприетарным системам, таким как VMware и Hyper-V.

Все эти функциональные возможности доступны с помощью скриптовой оболочки под названием guestfish или интерактивной оболочки восстановления virt-rescue.

libguestfs – это библиотека C, которая может быть связана с программами управления C и C ++ и имеет привязки для более десятка других языков программирования. Используя модуль FUSE, можно монтировать гостевые файловые системы на хосте. libguestfs постоянно совершенствуется с 2009 года. Сотни проектов используют libguestfs сегодня [10].

Данная библиотека является упрощенным вариантом обращения к образу системы libvirt. На рисунке 7 наиболее полно и развернуто представлена концепция виртуализации образа под управлением libvirt.

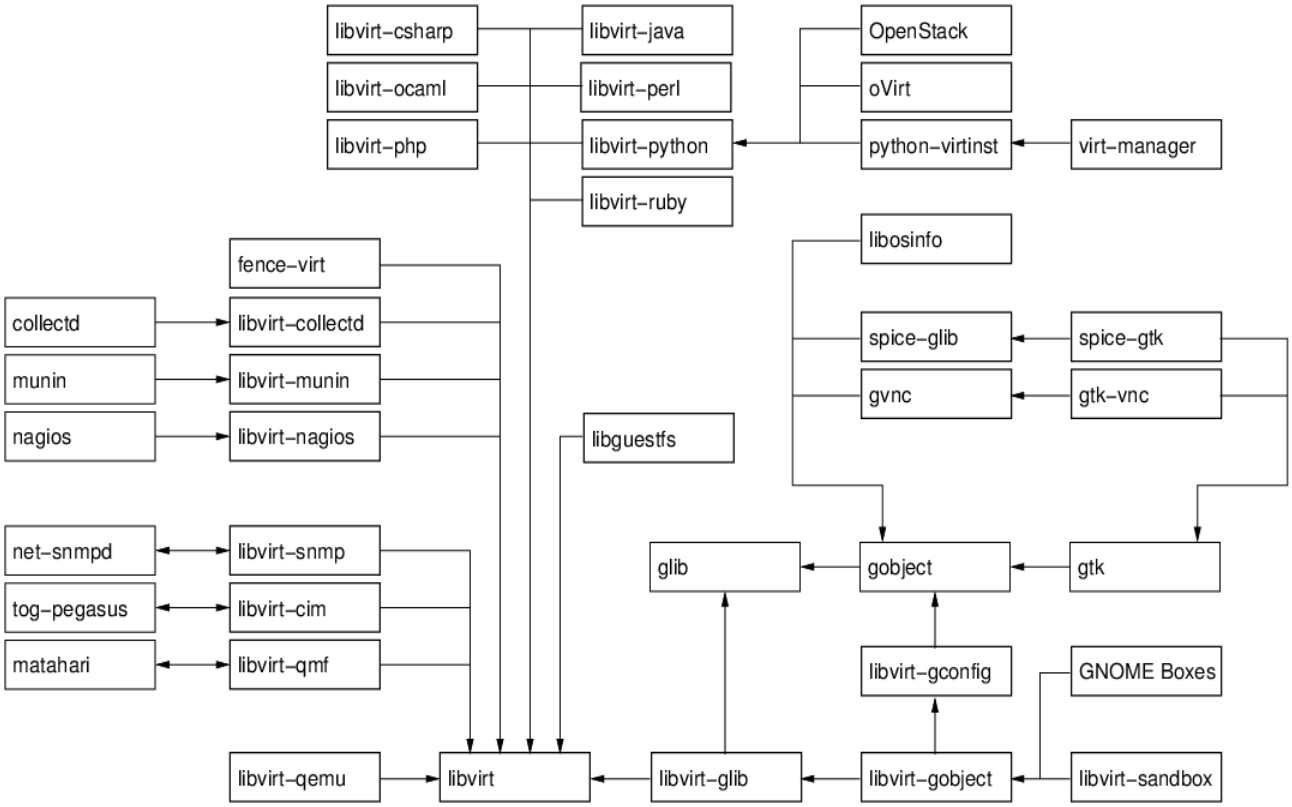


Рисунок 7 – Концепция построения дерева виртуализации образа

Данная концепция включает в себя управление ядром виртуальной машины (KVM). Гипервизор KVM представляет из себя загружаемый модуль ядра Linux, который предназначен для обеспечения виртуализации на платформе Linux x86.

Сам по себе KVM не выполняет эмуляции. Вместо этого программа, работающая в пространстве пользователя, использует интерфейс «/dev/kvm» для настройки адресного пространства гостя виртуальной машины, через него же эмулирует устройства ввода-вывода и видеоадаптер.

KVM позволяет виртуальным машинам использовать не модифицированные образы дисков VBox, VMware и других, содержащие операционные системы [11].

Именно KVM позволяет виртуализировать дисковое пространство исходной системы. Благодаря KVM имеется возможность использовать VDI и VMDK на основной машине без их непосредственного запуска.

Управление KVM происходит внутри библиотеки libvirt. На рисунке 9 представлена диаграмма классов виртуализации libvirt и внешних компонентов KVM и гипервизора Xen.

«Xen – кроссплатформенный гипервизор, разработанный в компьютерной лаборатории Кембриджского университета и распространяемый на условиях лицензии GPL. Основные особенности: поддержка режима паравиртуализации помимо аппаратной виртуализации, минимальность кода самого гипервизора за счёт выноса максимального количества компонентов за пределы гипервизора» [12]. Именно благодаря гипервизору Xen доступна паравиртуализация, то есть расслоение машины по процессам и эмуляция работы непосредственно необходимых. На нем базируется платформа Docker, которая позволяет удобно управлять необходимыми процессами. Структурная схема работы системы паравиртуализации Xen представлен на рисунке 8.

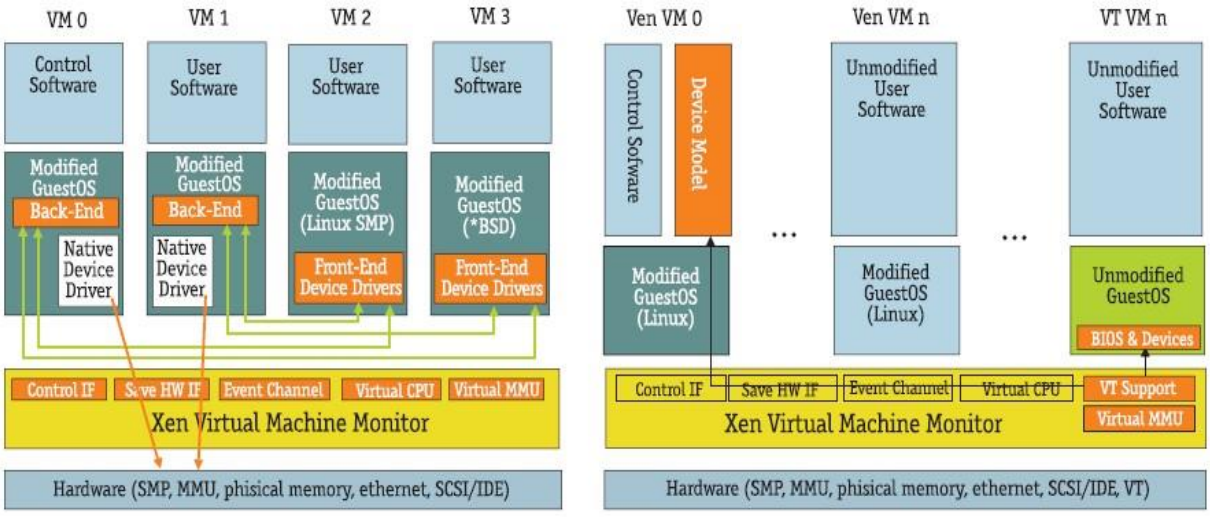


Рисунок 8 – Структурная схема выполнения процессов гипервизора Xen

На схеме видны основные компоненты работы гипервизора для паравиртуализации.

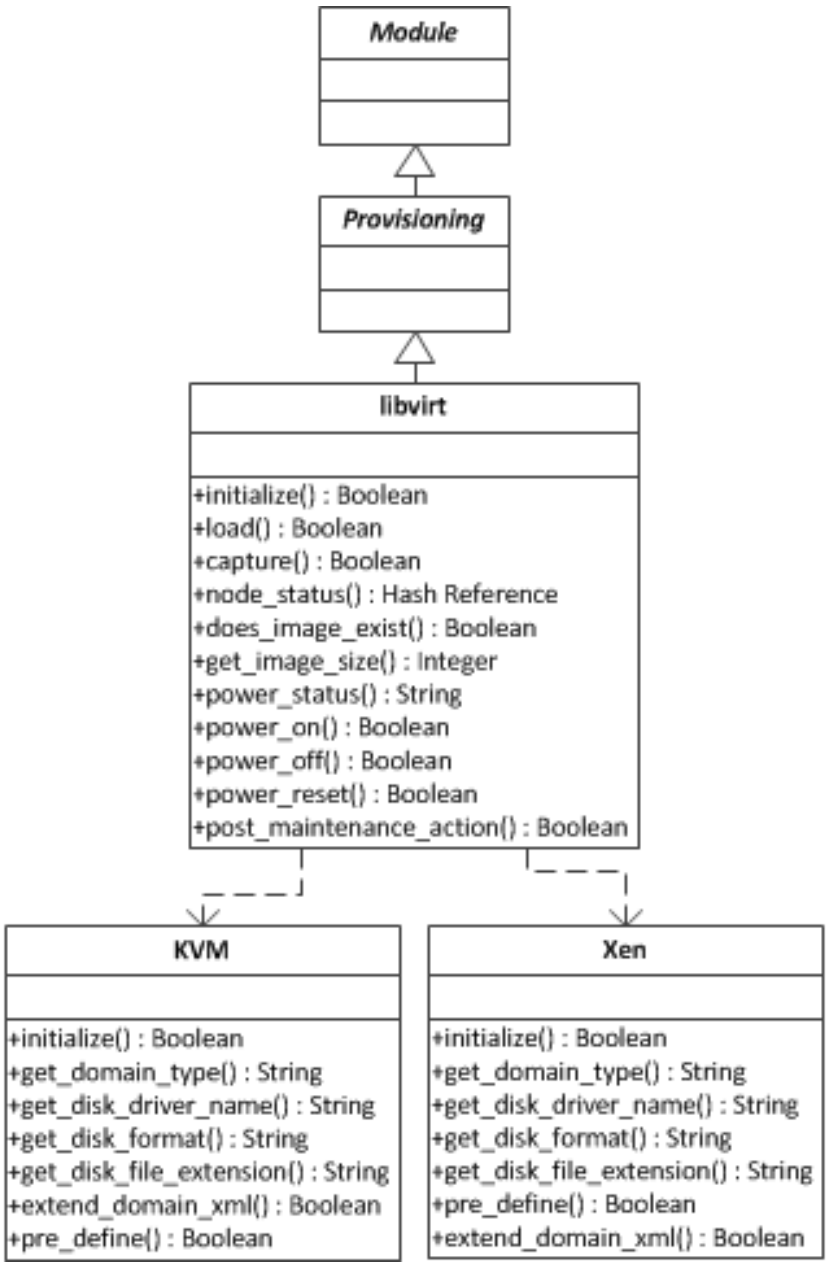


Рисунок 9 – Диаграмма классов виртуализации

Данная диаграмма показывает основные принципы любой виртуализации. Главным отличительным подходом со стороны Docker является объединенное использование библиотек KVM и Xen для одновременного эмулирования процессов и поддержания структурности отдельной операционной системы. Благодаря этому создается определенная прослойка. Получается так, что Docker эмулирует операционную систему в контейнере, но это не ограничивает его в использовании ресурсов основной (хостовой) машины.

2.2 Проектирование проверки требований используемой операционной системы

Для определения текущей операционной системы используется специальный файл, находящийся по пути «/etc/issue». Он используется по всех системах UNIX, поэтому определить подходящее семейство Linux не составит труда. В данном файле имеется название ОС и символьные константы для переноса каретки. Пример вывода файла для Ubuntu представлен на рисунке 10.

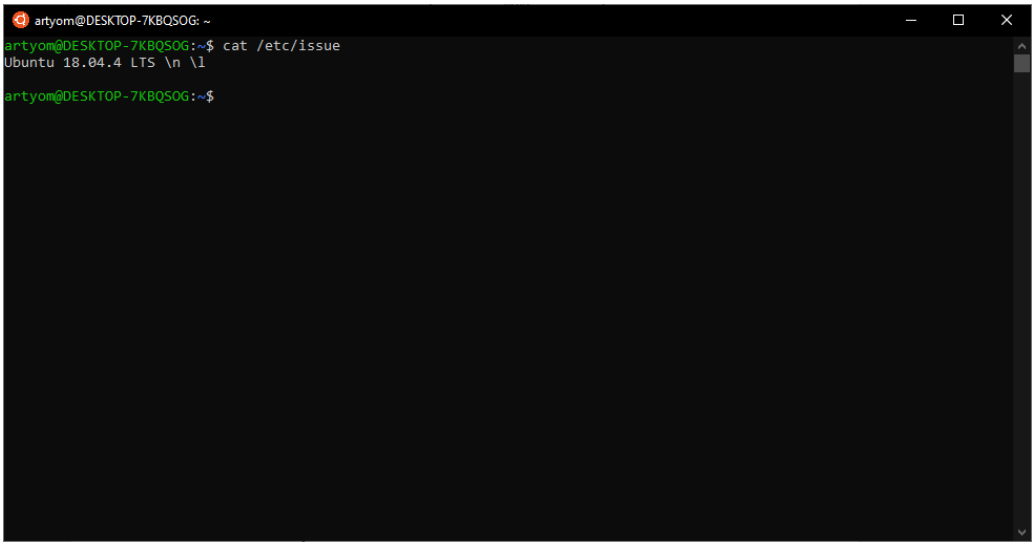


Рисунок 10 – Вывод файла issue в Ubuntu 18.04 LTS

Пример вывода файла для Debian представлен на рисунке 11.

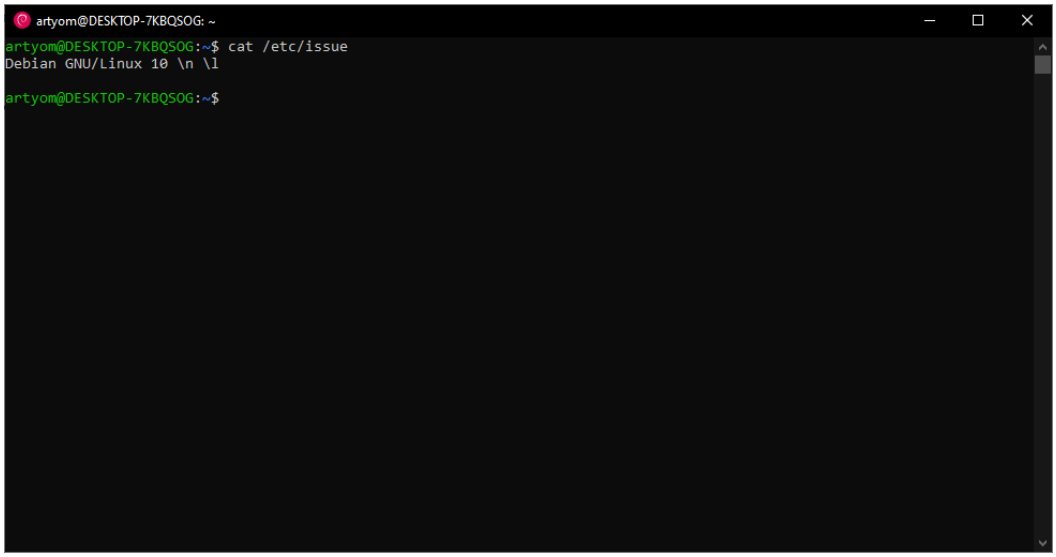
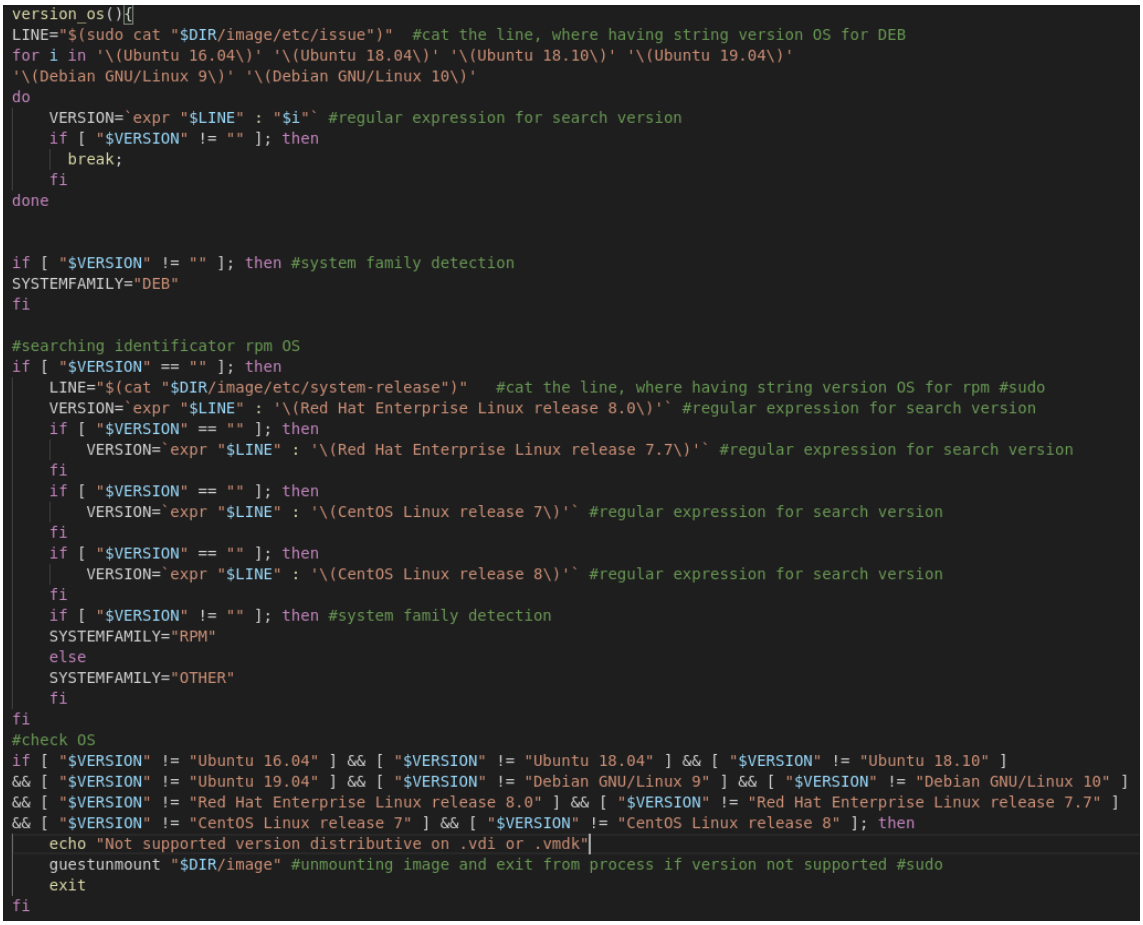


Рисунок 11 – Вывод файла issue в Debian 10

Для определения операционной системы достаточно узнать первые два компонента в данном файле. Пример определения ОС в Bash представлен на рисунке 12.



```
version_os(){
LINE="$(sudo cat "$DIR/image/etc/issue")"  #cat the line, where having string version OS for DEB
for i in '\(Ubuntu 16.04\)' '\(Ubuntu 18.04\)' '\(Ubuntu 18.10\)' '\(Ubuntu 19.04\)'
'\(Debian GNU/Linux 9\)' '\(Debian GNU/Linux 10\)'
do
    VERSION=`expr "$LINE" : "$i"` #regular expression for search version
    if [ "$VERSION" != "" ]; then
      break;
    fi
done


if [ "$VERSION" != "" ]; then #system family detection
SYSTEMFAMILY="DEB"
fi

#searching identificator rpm OS
if [ "$VERSION" == "" ]; then
    LINE="$(cat "$DIR/image/etc/system-release")"   #cat the line, where having string version OS for rpm #sudo
    VERSION=`expr "$LINE" : '\(Red Hat Enterprise Linux release 8.0\)'` #regular expression for search version
    if [ "$VERSION" == "" ]; then
        VERSION=`expr "$LINE" : '\(Red Hat Enterprise Linux release 7.7\)'` #regular expression for search version
    fi
    if [ "$VERSION" == "" ]; then
        VERSION=`expr "$LINE" : '\(CentOS Linux release 7\)'` #regular expression for search version
    fi
    if [ "$VERSION" == "" ]; then
        VERSION=`expr "$LINE" : '\(CentOS Linux release 8\)'` #regular expression for search version
    fi
    if [ "$VERSION" != "" ]; then #system family detection
    SYSTEMFAMILY="RPM"
    else
    SYSTEMFAMILY="OTHER"
    fi
fi
#check OS
if [ "$VERSION" != "Ubuntu 16.04" ] && [ "$VERSION" != "Ubuntu 18.04" ] && [ "$VERSION" != "Ubuntu 18.10" ]
&& [ "$VERSION" != "Ubuntu 19.04" ] && [ "$VERSION" != "Debian GNU/Linux 9" ] && [ "$VERSION" != "Debian GNU/Linux 10" ]
&& [ "$VERSION" != "Red Hat Enterprise Linux release 8.0" ] && [ "$VERSION" != "Red Hat Enterprise Linux release 7.7" ]
&& [ "$VERSION" != "CentOS Linux release 7" ] && [ "$VERSION" != "CentOS Linux release 8" ]; then
    echo "Not supported version distributive on .vdi or .vmdk"
    guestunmount "$DIR/image" #unmounting image and exit from process if version not supported #sudo
    exit
fi
```

Рисунок 12 – Функция определения ОС в Bash

Таким образом, можно определить дистрибутив Linux системы, но данное значение может быть изменено (обычно данная практика используется для корпоративных решений), поэтому имеется дополнительная проверка на определение пакетного менеджера. У всех дистрибутивов, рассматриваемых в задании, используется пакетный менеджер apt.

2.3 Проектирование генерации Dockerfile для управления платформой Docker

Dockerfile – это сценарий, который состоит из последовательности команд и аргументов, необходимых для создания образа. Такие сценарии упрощают развёртывание и процесс подготовки приложения к запуску.

Сначала Dockerfile определяет образ, на основе которого будет происходить сборка. Затем идёт ряд методов, команд и аргументов, которые создадут новый образ. Содержимое Dockerfile передаётся демону Docker для сборки образа.

В данной задаче самым простым решением для сборки контейнера является Dockerfile, так как его просто настроить и разбить на разные компоненты программы. На рисунке 14 представлена диаграмма компонентов программы. Она показывает модульность самой программы, т.е. можно добавить любой программный компонент при необходимости.

Данная диаграмма показывает, что любое программное средство, установленное в образе виртуальной машины, можно описать новым «.sh» компонентом. Пример компонента apache2.sh представлен на рисунке 13.

```
# !/bin/bash

# This configuration file for a specific program - Apache

IMPROVEDNAME="Apache"
LOWERCASENAME="apache"
#VERSION_STR="`grep -REHn --colour=auto "^Package: apache2" --include='image' | head -1 | cut -d ":" -f2`" TODO
VERSION="`$DIR/image/usr/sbin/apache2 -v | head -1 | cut -d " " -f3 | cut -d "/" -f2`"
MAINCONFIG="$DIR/image/etc/apache2/apache2.conf"
if ! [ -d "$OUTPUT_DIR/configs/Apache" ]; then #create a directory if it does not exist
    mkdir "$OUTPUT_DIR/configs/Apache";
fi
if ! [ -d "$OUTPUT_DIR/data/Apache" ]; then #create a directory if it does not exist
    mkdir "$OUTPUT_DIR/data/Apache";
fi
if ! [ -d "$OUTPUT_DIR/data/Apache/www" ]; then #create a directory if it does not exist
    mkdir "$OUTPUT_DIR/data/Apache/www";
fi

PATHSYSTEM=`grep -iRI "DocumentRoot " "$DIR/image/etc/apache2/sites-enabled/" | cut -f2 | cut -d " " -f2 | sed 's/\;\S*//'`

for i in $PATHSYSTEM; do
`cp -r "$DIR/image$i" "$OUTPUT_DIR/data/Apache/www"` #copying all file structure
done

`cp -r "$DIR/image/etc/apache2/." "$OUTPUT_DIR/configs/Apache"` #copying all file structure
#`cp -r "$DIR/image/var/www/." "$OUTPUT_DIR/data/Apache/www"` #copying all file structure
DBDIRECTORY="`grep -r -E "^<Directory" "$MAINCONFIG" | tail -1 | cut -d " " -f2 | rev | cut -c 2- | rev`"
PORT="`grep -r -E "^Listen" "$DIR/image/etc/apache2/ports.conf" | awk '{print $(NF)}'`"

cat >"$DIR/docker/$NAME2/Dockerfile" << EOF
FROM $SYSTEMNAME:$SYSTEMVERSION
ENV DEBIAN_FRONTEND=noninteractive
RUN apt update
RUN apt upgrade -y
RUN mkdir /var/run/apache2
RUN apt install apache2 -y
RUN /bin/bash -c "source /etc/apache2/envvars"
RUN apt install php -y
#ENTRYPOINT [ "apache2" ]
#COPY $OUTPUT_DIR/configs/Apache/ /etc/apache2/
EOF
docker build -t "$LOWERCASENAME" "$DIR/docker/$NAME2"
```

Рисунок 13 – Компонент apache2.sh

Пример компонента mariadb.sh представлен на рисунке 14.

```
# !/bin/bash

# This configuration file for a specific program - MariaDB

IMPROVEDNAME="MariaDB"
LOWERCASENAME="mariadb"
VERSION="10.4" #connect to database unstable
MAINCONFIG="$DIR/image/etc/mysql/my.cnf"
if ! [ -d "$OUTPUT_DIR/configs/MariaDB" ]; then #create a directory if it does not exist
  mkdir "$OUTPUT_DIR/configs/MariaDB";
fi
`cp $(echo "$MAINCONFIG" | awk '{print $(NF)}' ) "$OUTPUT_DIR/configs/MariaDB/my.cnf"`
DBDIRECTORY="`grep -r datadir "$DIR/image/etc/mysql" | head -1 | awk '{print $(NF)}'`"
#DBDIRECTORYCP="`grep -r datadir "$DIR/image/etc/mysql" | head -1 | awk '{print $(NF)}'`"
PORT="`grep -r -E "^port" "$DIR/image/etc/mysql" | head -1 | awk '{print $(NF)}'`"

if ! [ -d "$OUTPUT_DIR/data/MariaDB/" ]; then #create a directory if it does not exist
  mkdir "$OUTPUT_DIR/data/MariaDB/";
fi
if ! [ -d "$OUTPUT_DIR/data/MariaDB/db" ]; then #create a directory if it does not exist
  mkdir "$OUTPUT_DIR/data/MariaDB/db";
fi
if ! [ -d "$OUTPUT_DIR/configs/MariaDB/etc" ]; then #create a directory if it does not exist
  mkdir "$OUTPUT_DIR/configs/MariaDB/etc";
fi
#if ! [ -d "$OUTPUT_DIR/configs/MariaDB/mysqld" ]; then #create a directory if it does not exist
#  mkdir "$OUTPUT_DIR/configs/MariaDB/mysqld";
#fi
`cp -r "$DIR/image/var/lib/mysql/." "$OUTPUT_DIR/data/MariaDB/db"` #copying all file structure
`cp -r "$DIR/image/etc/mysql/." "$OUTPUT_DIR/configs/MariaDB/etc"` #copying all file structure
#`cp -r "$DIR/image/var/run/mysqld/." "$OUTPUT_DIR/configs/MariaDB/mysqld"` #copying all file structure

cat >"$DIR/docker/$NAME2/Dockerfile" << EOF
FROM mariadb:latest
ENV DEBIAN_FRONTEND=noninteractive
# RUN echo "deb http://archive.ubuntu.com/ubuntu/ bionic main restricted universe multiverse" >> /etc/apt/sources.list
# #RUN echo "deb [arch=ppc64el,amd64,arm64] http://mariadb.mirror.liquidtelecom.com/repo/10.4/ubuntu bionic main" >> /etc/apt/sources.list
RUN apt update
RUN apt upgrade -y
# RUN apt install mariadb-client -y
# RUN apt install mariadb-server -y
# COPY $OUTPUT_DIR/configs/MariaDB/mariadb.cnf /etc/mysql/mariadb.cnf
EOF
docker build -t "$LOWERCASENAME" "$DIR/docker/$NAME2"
```

Рисунок 14 – Компонент mariadb.sh

Сам Dockerfile формируется в коде путем дописывания в зависимости от программы, найденной на образе исходной системы. Пример готового Dockerfile представлен на рисунке 15.

Данный файл сохраняется по корневому пути. Пользователь, после автоматического переноса, может создать чистый образ виртуальной машины, т.е. без пользовательских данных.

```
FROM ubuntu:18.04
ENV DEBIAN_FRONTEND=noninteractive
WORKDIR /src
RUN echo "deb http://archive.ubuntu.com/ubuntu/ bionic main restricted universe multiverse" >> /etc/apt/sources.list
RUN echo "deb http://archive.ubuntu.com/ubuntu/ bionic-updates main restricted universe multiverse" >> /etc/apt/sources.list
RUN echo "deb http://archive.ubuntu.com/ubuntu/ bionic-security main restricted universe multiverse" >> /etc/apt/sources.list
RUN apt update
#RUN apt-get upgrade -y
RUN apt install libguestfs-tools -y
COPY ./vm2docker.sh /src/
COPY ./apache2.sh /src/
COPY ./iptables.sh /src/
COPY ./java.sh /src/
COPY ./mariadb.sh /src/
COPY ./mysql.sh /src/
COPY ./nginx.sh /src/
COPY ./php.sh /src/
COPY ./postgresql.sh /src/
COPY ./redis.sh /src/
RUN chmod +x /src/vm2docker.sh
ENTRYPOINT ["bash", "/src/vm2docker.sh" ]

#docker build -t vm2docker -- build docker image
```

Рисунок 15 – готовый Dockerfile

Данный подход является удачным решением, т.к. пользователь может одной командой создать контейнер с установленными программными продуктами, нужными ему. Для написания необходимо расширения достаточно знать название пакета, данные, необходимые для переноса.

Таким образом, Dockerfile является удобным и универсальным компонентом для создания Docker контейнеров системы. Каждая машина может создаваться с пустым набором данных, так и переноситься со всеми данными, как и на виртуальной машине.

3 Реализация программного средства для автоматического конвертирования образов систем

Программные средства были реализованы на скриптовых языках Bash и Python в средах Visual Studio Code и JetBrains PyCharm соответственно. Исходный код представлен в приложении Б. Диаграмма компонентов представлена на рисунке 16.

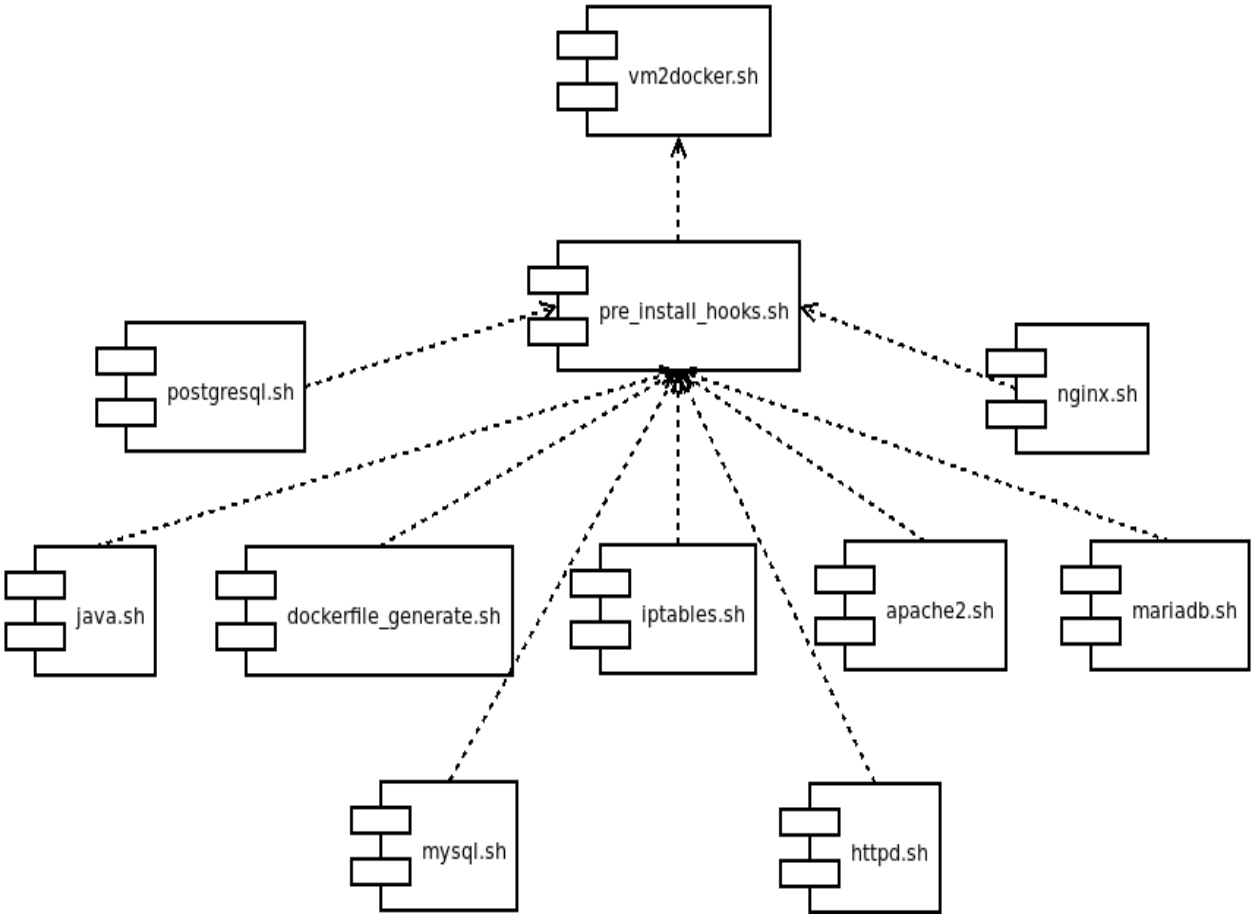


Рисунок 16 – Диаграмма компонентов программы

На данной диаграмме имеются зависимости между компонентами. Можно заметить, что использование такого подхода к построению компонентов, поможет в дальнейшем усовершенствовать программное средство путем добавления новых программных компонент. В зависимости от поставленной задачи переноса программных средств, можно добавить необходимое. Описание компонентов представлено в таблице 1.

Таблица 1 – Описание компонентов проекта

Название	Описание
vm2docker.sh	Данный файл является основным в программе, куда передается вся информация
pre_install_hooks.sh	Данный файл необходим для установки всех необходимых библиотек на хостовой машине
postgresql.sh	Скрипт конфигурации PostgreSQL
nginx.sh	Скрипт конфигурации Nginx
Название	Описание
java.sh	Скрипт конфигурации Java JRE (JDK)
iptables.sh	Скрипт конфигурации IpTables
apache2.sh	Скрипт конфигурации Apache (Tomcat)
mariadb.sh	Скрипт конфигурации MariaDB
mysql.sh	Скрипт конфигурации MySQL
httpd.sh	Скрипт конфигурации Apache для RHEL Linux машин
dockerfile_generate.sh	Скрипт для создания Dockerfile

Диаграмма развертывания на примере одного из компонентов представлена на рисунке 17. На данной диаграмме видны отношения и взаимодействия работы Docker с хост-машиной и средой выполнения Linux. В конкретном случае, для данного взаимодействия используется терминал, который производит взаимодействие между средами исполнения. Диаграмма показывает принцип работы среды выполнения Docker и ее взаимодействия со средой выполнения GNU/Bash и хостовой машиной. Docker создает внутри своей среды выполнения не новые среды, а новые машины, которые являются частью хост-машины. Главным отличием от виртуальных машин является создание необособленных новых машин, связанных вместе под одной средой выполнения.

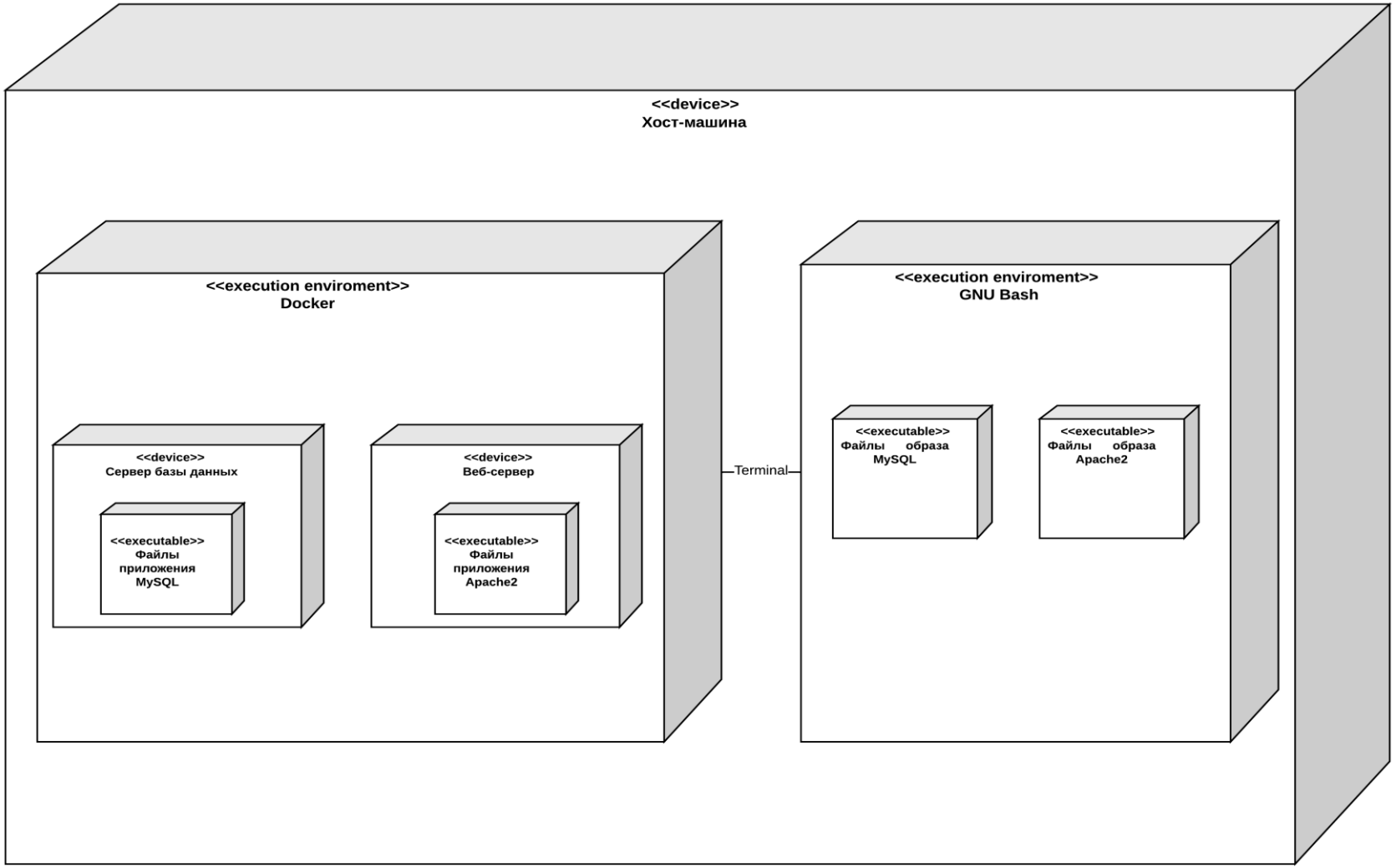


Рисунок 17 – Диаграмма развертывания программного средства

Таким образом, были разработаны программные средства, реализованные на скриптовых языках Bash и Python в средах Visual Studio Code и JetBrains PyCharm соответственно. Были разработаны диаграмма компонентов с подробным описанием каждого модуля программного средства и диаграмма развертывания, показывающая смысл работы контейнеризации в платформе Docker.

4 Тестирование программного средства

Для проверки работоспособности программных средств будем использовать функциональное тестирование [13].

Таблица 2 – Функциональное тестирование

Вариант использования	Параметры теста	Результат
Конвертировать в Docker контейнер	Использование образа диска Ubuntu 18.04	Тест пройден. Образ Docker успешно получен (рисунок 18)
Конвертировать в Docker контейнер с сохранением в YAML файл	Использование образа диска Debian 9 с установленными программами Apache 2.4.25, Nginx 1.14.0, PHP 1.14.0	Тест пройден. Результаты программы выведены в YAML файл (рисунок 19)
Проверить образ диска	Проверить не валидный образ диска	Тест пройден. Образ успешно проверен и выдан результат (рисунок 20)

На рисунке 18 видно выполнение полного цикла работы и создание новых контейнеров из образов системы.

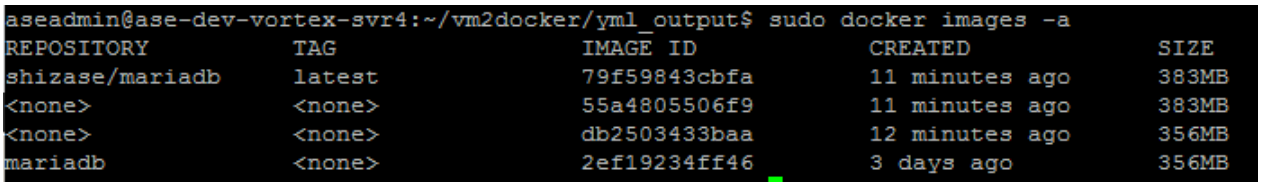


Рисунок 18 – Результат тестирования сохранения образа системы

На рисунке 19 представлен YAML файл, полученный в ходе выполнения программы.

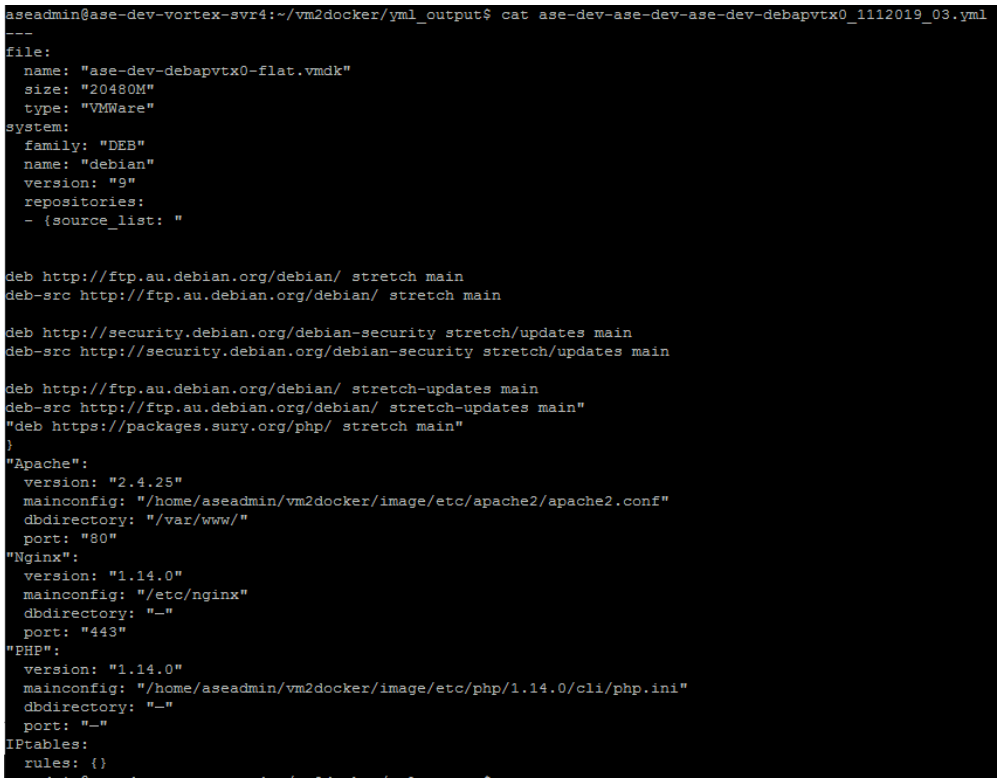


Рисунок 19 – Результат конвертирования образа и сохранения YAML

На рисунке 20 изображен результат при использовании не валидного образа системы.

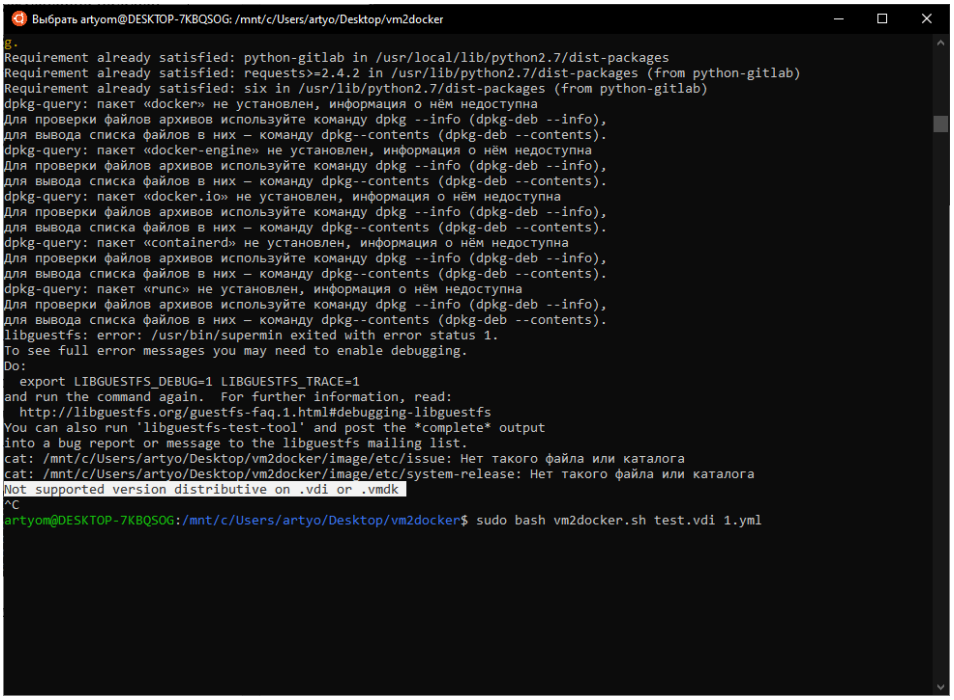


Рисунок 20 – Результат тестирования не валидного образа системы

Тестирование выполнено в полном объеме. Выявленных ошибок обнаружено не было.

4.1 Рефакторинг кода

Для рефакторинга Bash скрипта используется специальная программа «shellcheck». Он помогает искать ошибки в текстах скриптов и выводить отчёты о проведённых проверках. Это позволяет повысить производительность работы и качество кода.

Проверка данной программой показала предложения для упрощения кода и 1 критическую ошибку, при которой возможен бесконечный цикл. Результат работы программы представлен на рисунках 21 и 22.

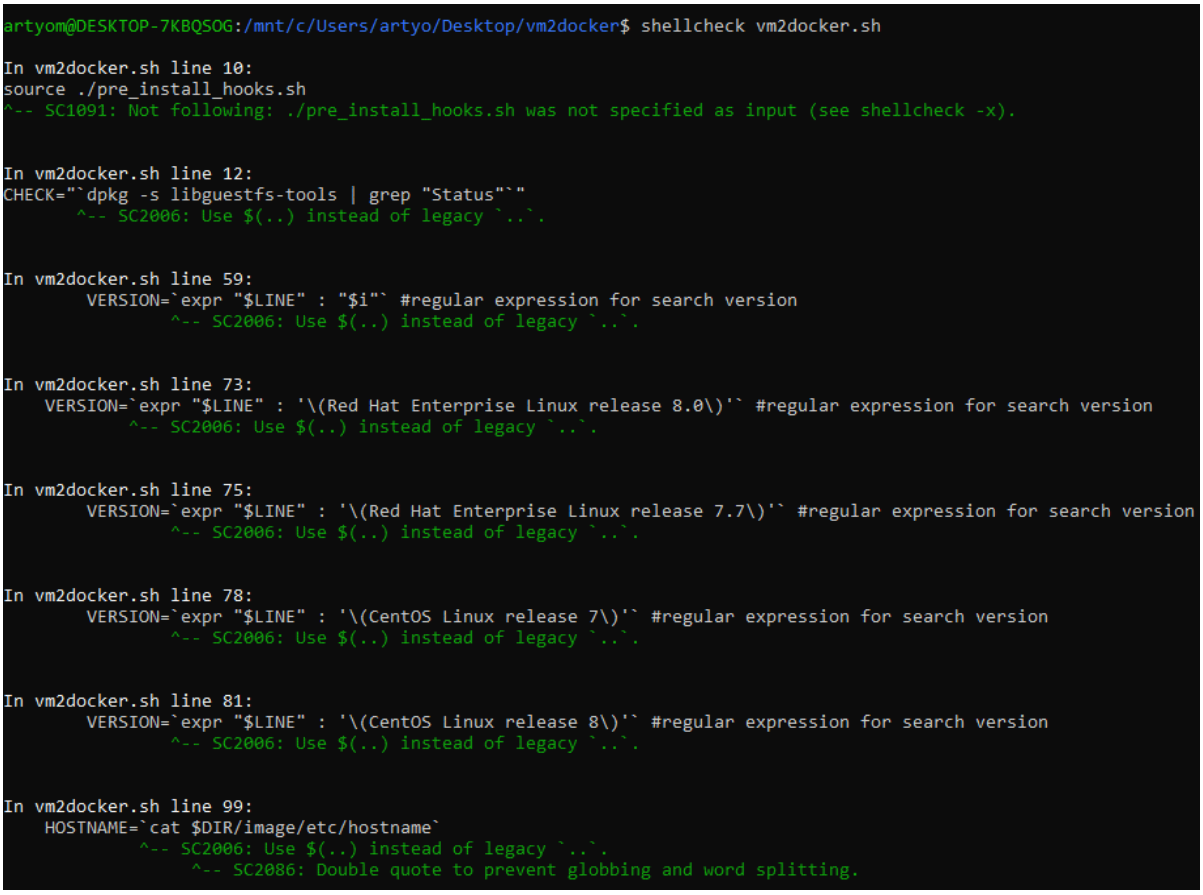


Рисунок 21 – Результат работы программы shellcheck (начало)

На изображении видно, что основным рефакторингом является исправление функций и приведение их к виду «$(..)». Данное исправление является верным, так как если потребуется дописать новое выполнение

команды, это не потребует введения новых двойных кавычек, тем самым это улучшает читаемость кода.

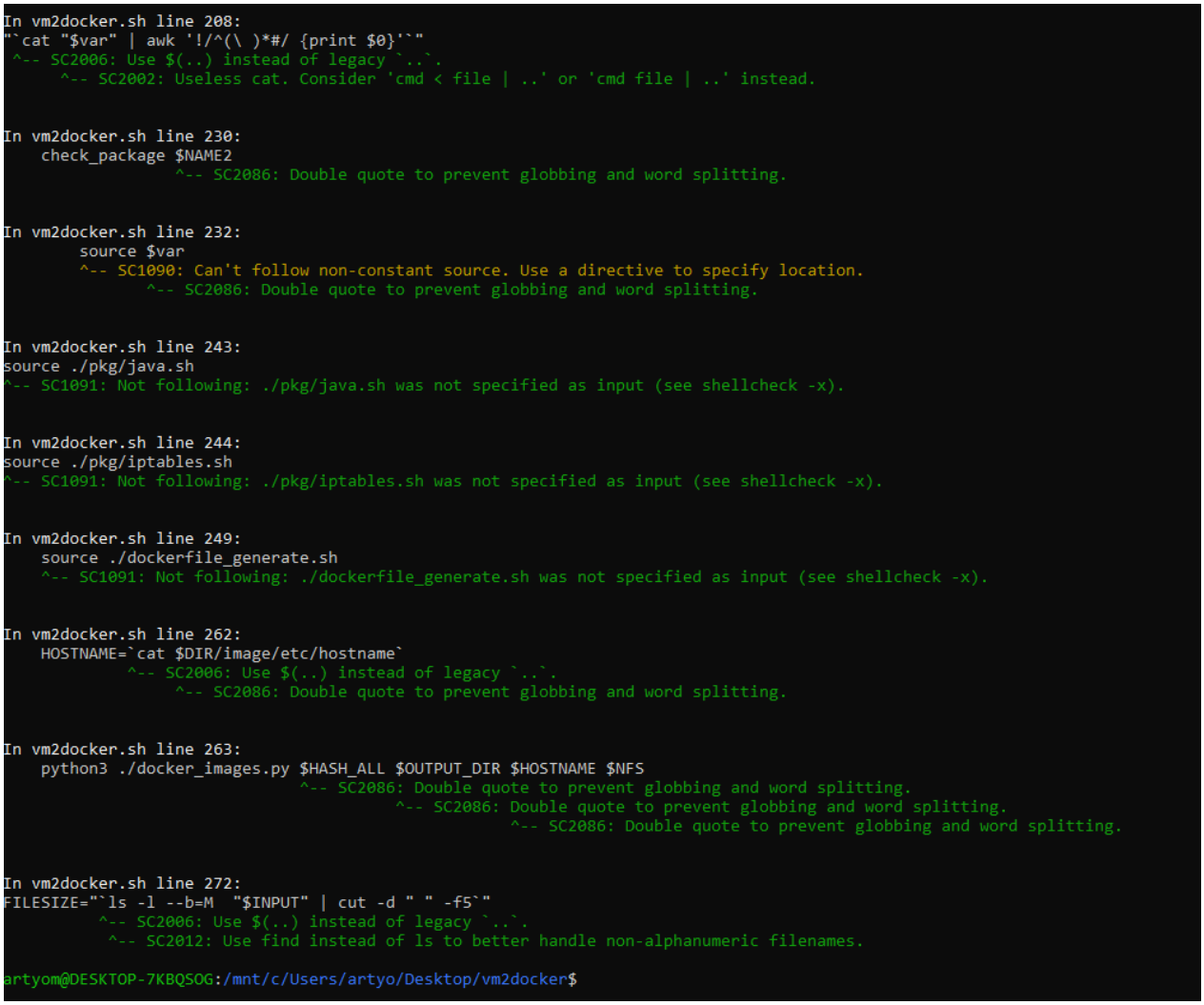


Рисунок 22 – Результат работы программы shellcheck (окончание)

В результате рефакторинга кода данным способом повысилась читаемость кода и удобство добавления нового функционала.

5 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета была выполнена при помощи программы для управления проектами Microsoft Project [14]. Проект был начат 13 апреля 2020 года. При планировании разработки были выделены ряд задач. Список задач и их описание представлено на рисунке 23.

		Начало проекта	0 дней	Пн 13.04.20	Пн 13.04.20		
2		**Анализ требований и постанавка задачи**	**9 дней**	**Вт 14.04.20**	**Пт 24.04.20**	**1**	
3		Анализ предметной области	1 день	Пн 20.04.20	Пн 20.04.20		Аналитик[30%]; Программист
4		Сравнение программ-аналогов	2 дней	Пн 20.04.20	Вт 21.04.20		Аналитик[70%]
5		Постановка задачи	1 день	Ср 22.04.20	Ср 22.04.20	3;4	Аналитик
6		Анализ функциональных требований	1 день	Чт 23.04.20	Чт 23.04.20	5	Аналитик; Программист
7		Анализ требований и постановка задачи средств завершено	0 дней	Чт 23.04.20	Чт 23.04.20	6	
8		**Проектирование программных средств**	**16 дней**	**Пн 20.04.20**	**Пн 11.05.20**	**2**	
9		Проектирование виртуализации дискового пространства исходного образа системы	5 дней	Пн 20.04.20	Пт 24.04.20		Аналитик
10		Проектирование провеки требований используемой операционной системы	4 дней	Вт 21.04.20	Пт 01.05.20		Программист[25%]
11		Проектирование генерации Dockerfile для управления платформой Docker	6 дней	Пн 04.05.20	Пн 11.05.20	9;10	Аналитик
12		Проектирование программных средств завершено	0 дней	Пн 11.05.20	Пн 11.05.20	11	
13		**Реализация**	**17 дней**	**Пн 11.05.20**	**Вт 02.06.20**	**8**	
14		Реализация виртуализации дискового пространства исходного образа системы	7 дней	Пн 11.05.20	Вт 19.05.20		Программист
15		Реализация проверки требований используемой операционной системы	7 дней	Ср 20.05.20	Чт 28.05.20	14	Программист[80%]
16		Реализация генерации Dockerfile для управлений пллатформой Docker	3 дней	Чт 28.05.20	Пн 01.06.20	14	Программист[20%]
17		Реализация завершена	0 дней	Пн 01.06.20	Пн 01.06.20	15;16	
18		**Тестирование**	**7 дней**	**Пн 01.06.20**	**Вт 09.06.20**	**13**	
19		Функциональное тестирование	4 дней	Пн 01.06.20	Чт 04.06.20		Программист; Тестировщик
20		Модульное тестирование	3 дней	Пт 05.06.20	Вт 09.06.20	19	Тестировщик
21		Тестирование завершено	0 дней	Пт 05.06.20	Пт 05.06.20	20	
22		Составление и оформление документации	3 дней	Ср 10.06.20	Пт 12.06.20	18	Аналитик
23		Конец проекта	0 дней	Сб 13.06.20	Сб 13.06.20	22	

Рисунок 23 – Список задач проекта

Для реализации проекта было выделено три трудовых ресурса: аналитик, программист и тестировщик. Описание ресурсов представлено на рисунке 24.

	Название ресурса	Тип	Единицы измерения материалс	Краткое названи	Группа	Макс. единиц	Стандартна ставка	Ставка сверхурочн	Затраты на использ.	Начисление	Базовый календарь	Код	авить новый стол
1	Аналитик	Трудовой		А		100%	130,00 ₽/ч	260,00 ₽/ч	0,00 ₽	Пропорциональ	Стандартный		
2	Программист	Трудовой		П		100%	208,00 ₽/ч	416,00 ₽/ч	0,00 ₽	Пропорциональ	Стандартный		
3	Тестировщик	Трудовой		Т		100%	168,00 ₽/ч	336,00 ₽/ч	0,00 ₽	Пропорциональ	Стандартный		

Рисунок 24 – Список ресурсов проекта

В результате выделения задач и необходимых ресурсов проекта была построена диаграмма Ганта (рисунок 25). На диаграмме Ганта представлены задачи продолжительность и взаимосвязи с между задачами в проекте. Также на диаграмме отображены даты начала и конца каждой задачи и назначенные ресурсы для выполнения задачи.

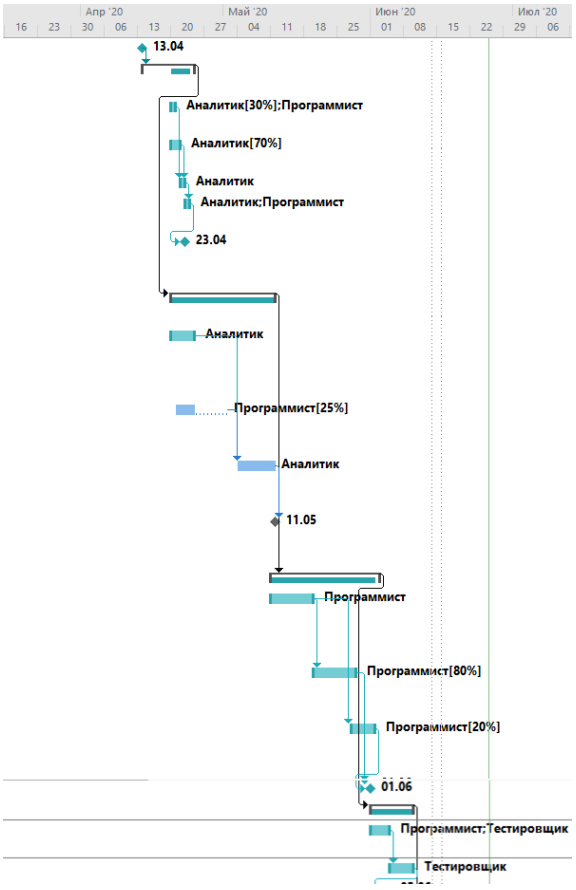


Рисунок 25 – Диаграмма Ганта

Сетевой график, составленный в Microsoft Project, представлен на рисунках 26 - 28.

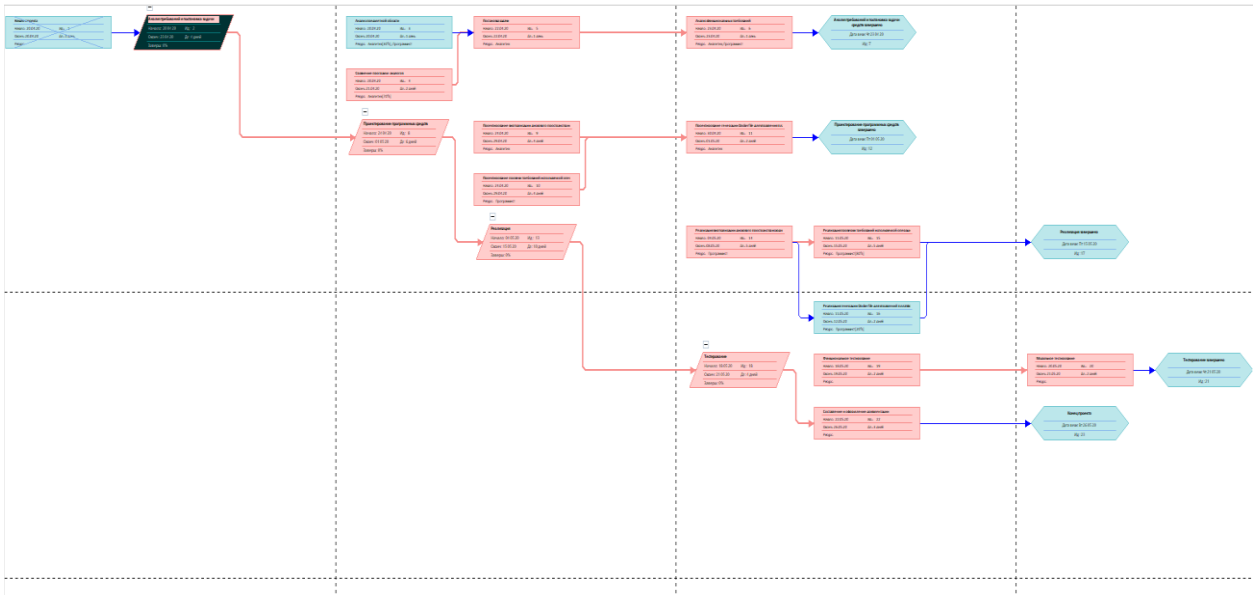


Рисунок 26 – Полный сетевой график

На первой части сетевого графика представлены основные этапы планирования разработки (рисунок 27).

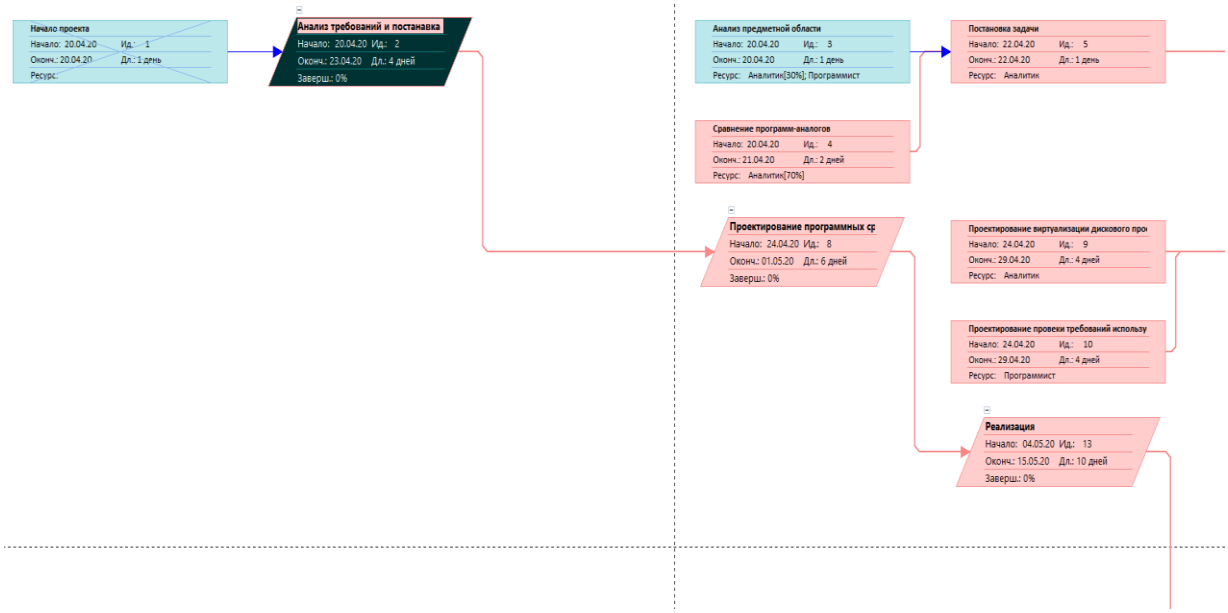


Рисунок 27 – Часть 1 сетевого графика

На второй части сетевого графика представлено проектирование и реализация программного средства.

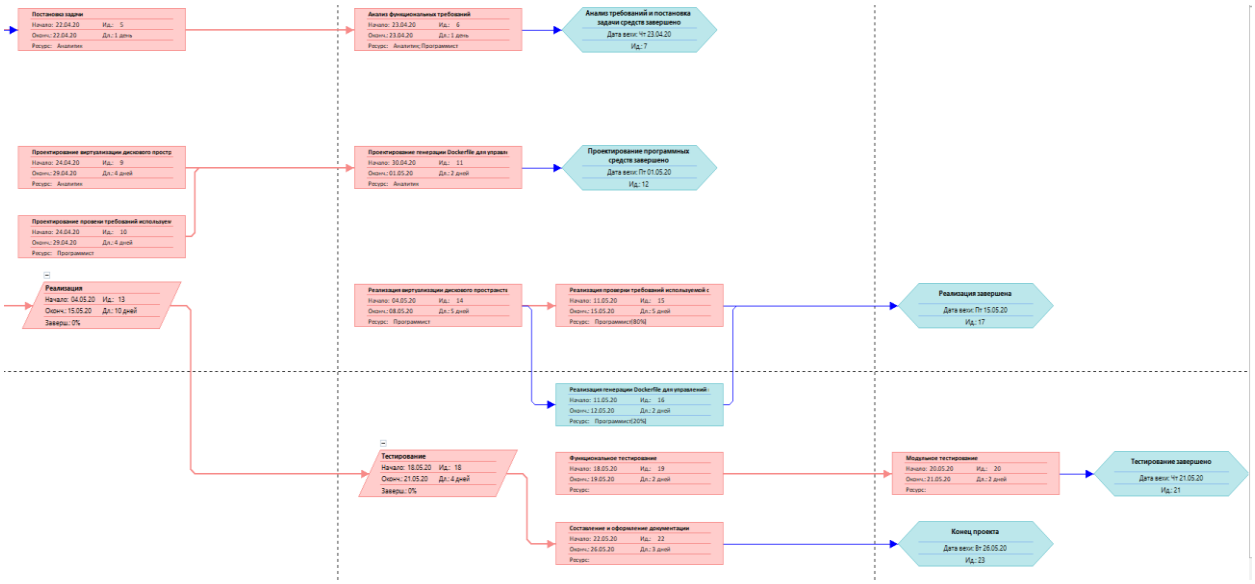


Рисунок 28 – Часть 2 сетевого графика

На рисунке 29 представлена гистограмма загруженности ресурсов.

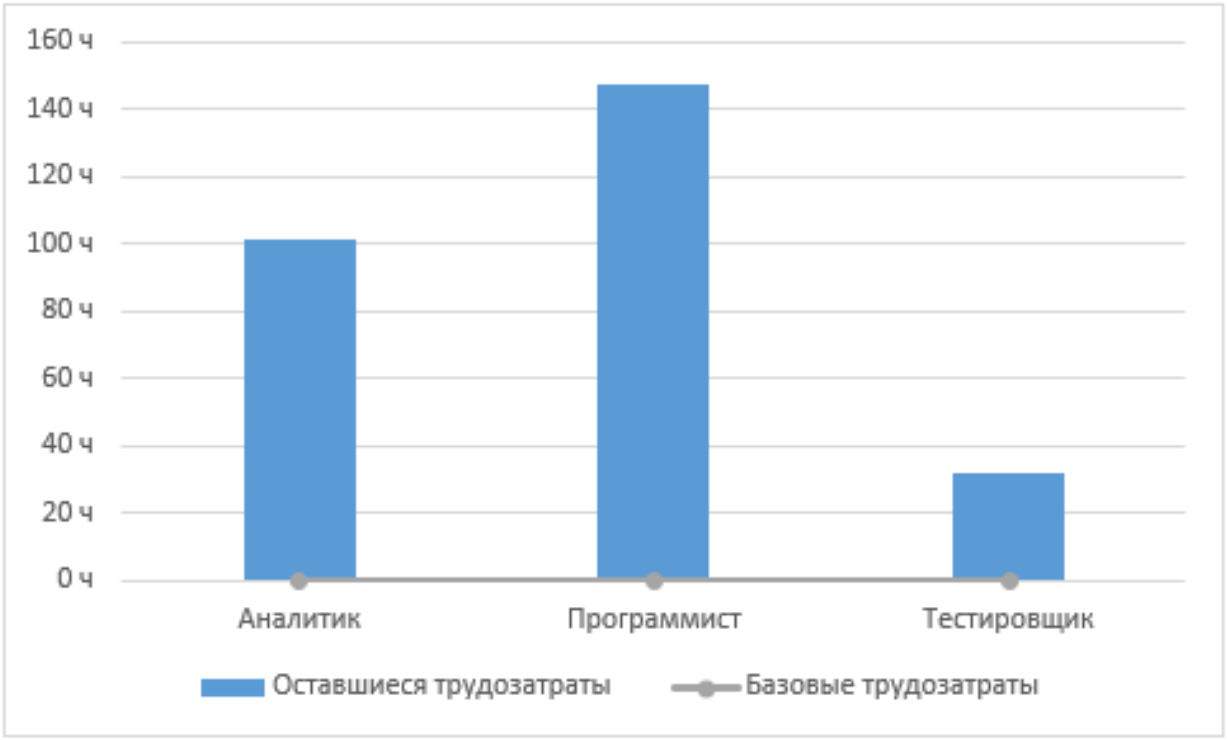


Рисунок 29 – Гистограмма загруженности ресурсов

Таким образом разработка проекта составит 27 полных рабочих дней с 20 апреля 2020 года по 26 мая 2020 года. Трудозатраты составят 280 человеко-часов. Оценка бюджета проекта составляет 38523 рубля.

Заключение

В ходе выполнения выпускной квалификационной работы разработаны программные средства для реализации автоматического конвертирования образов систем.

Был проведен анализ программ, доступных для автоматического переноса в контейнеры и их возможная совместимость. В рамках работы был проведен сравнительный анализ программных продуктов-аналогов. Были исследованы следующие продукты-аналоги. Каждый продукт-аналог был охарактеризован, были приведены его преимущества и недостатки.

На основе полученной информации была сформулирована задача на разработку программного средства, определены функциональные и нефункциональные требования к приложению.

Для решения данной задачи были выделены основные этапы построения данного программного средства:

– виртуализация дискового пространства исходного образа системы;

– проверка требований используемой операционной системы;

– генерация Dockerfile для управления платформой Docker;

Данные о проектировании представлены в виде диаграмм и сегментов кода, удовлетворяющим концепции разработки. С целью простоты понимания и дальнейшего усовершенствования кода была использована библиотека shellcheck.

В отличие от известных, разработанное программное средство обеспечивает удобство и полную универсальность в переносе связанных многомодульных систем. Данное приложение претендует на роль уникального на рынке программных средств.

Разработка программных средств проводилась на языках программирования Bash и Python в средах Visual Studio Code и JetBrains PyCharm в операционной системе Windows. Тестирование и отладка

проводилась на основе Docker в операционной системе Ubuntu Server 18.04 LTS семейства Linux. Все тесты были успешно пройдены и в результате тестирования не выявлено ошибок.

В ходе работы было проведено планирование и оценка бюджета разработки при помощи программы для управления проектами Microsoft Project. Трудозатраты составили 280 человеко-часов. Затраты на проект составили 38523 рубля. Время разработки составило 27 рабочих дней.

Таким образом, в результате выполнения квалификационной работы были разработаны программные средства для реализации автоматического конвертирования образов систем, которые удовлетворяют требованиям технического задания.

Список использованных источников

1 How the go programming language helps Docker and the container ecosystem // TheNewStack [Электронный ресурс]. URL: https://thenewstack.io/go-programming-language-helps-docker-container-ecosystem (дата обращения: 22.05.2020).

2 Понимая Docker // Хабр [Электронный ресурс]. URL: https://habr.com/ru/post/253877 (дата обращения: 22.05.2020).

3 Главная страница Docker Hub // Docker Hub [Электронный ресурс]. URL: https://hub.docker.com (дата обращения: 22.05.2020).

4 Marks M. Image2Docker: A New Tool for Prototyping Windows VM Conversions // Docker Blog [Электронный ресурс]. URL: https://www.docker.com/blog/image2docker-prototyping-windows-vm-conversions (дата обращения: 23.05.2020).

5 Image2Docker // GitHub [Электронный ресурс]. URL: https://github.com/docker-archive/communitytools-image2docker-win (дата обращения: 23.04.2020).

6 Главная страница Rancher // Rancher [Электронный ресурс]. URL: https://rancher.com (дата обращения: 23.05.2020).

7 RancherVM // GitHub [Электронный ресурс]. URL: https://github.com/rancher/vm (дата обращения: 23.05.2020).

8 RancherVM Issues // GitHub [Электронный ресурс]. URL: https://github.com/rancher/vm/issues/117 (дата обращения: 1.06.2020).

9 Химонин Ю.И. Сбор и анализ требований к программному продукт. // Московское отделение Project Management Institute [Электронный ресурс]. URL: http://old.pmi.ru/profes/Software_Requirements_Khimonin.pdf (дата обращения: 1.06.2020).

10 Главная страница libguestfs // libguestfs [Электронный ресурс]. URL: http://libguestfs.org (дата обращения: 12.06.2020).

11 Узнайте о виртуальной машине ядра Linux (KVM) // IBM [Электронный ресурс]. URL: https://www.ibm.com/developerworks/ru/library/l-linux-kvm (дата обращения: 15.06.2020).

12 Чубин И. Xen // Xgu [Электронный ресурс]. URL: http://xgu.ru/wiki/Xen (дата обращения: 15.06.2020).

13 Категории программных тестов // Хабр [Электронный ресурс]. https://habr.com/ru/post/64874 (дата обращения 14.06.2020)

14 Джонсон, Т. Microsoft Project 2016. Шаг за шагом. / Т. Джонсон, К. Четфилд – Москва: КТК Галактика, 2016. – 594 с.

Приложение А

ГЛОССАРИЙ

(справочное)

Виртуальная машина – система, как аппаратная, так и программная, которая может эмулировать программное обеспечение определенной платформы.

Семейство операционных систем DEB – операционные системы семейства GNU/Linux, основанные на ядре управления Debian.

Хост-машина – машина, на которой запускается виртуальная система.

Docker контейнер – это изолированная от основной операционной системы совокупность процессов.

Dockerfile – это набор инструкций, выполняемых над образом системы, для построения Docker контейнера.

Linux – семейство Unix-подобных операционных систем на базе ядра Linux.

Libvirt – библиотека для виртуализации процессов на хостовой машине.

VDI, VMDK – форматы файлов, разрабатываемые Oracle и VMware соответственно, для использования в качестве образа диска своих виртуальных машин.

Xen – кроссплатформенный гипервизор, разработанный в компьютерной лаборатории Кембриджского университета и распространяемый на условиях лицензии GPL.

YAML – «дружественный» формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода типичных структур данных многих языков программирования.

Приложение Б

ЛИСТИНГ ПРОГРАММЫ

(рекомендуемое)

Листинг Б.1 – Файл vm2docker.sh

```
# !/bin/bash


# This is the main control script to run the entire project.

set +e

#Checking whether or not guestmount library installed and stores input
parameters
check_parametrs(){
source ./pre_install_hooks.sh
#check package for guestmount
CHECK="`dpkg -s libguestfs-tools | grep "Status"`"
if ! [ -n "$CHECK" ];
then apt install libguestfs-tools -y #library for mounting image #sudo
fi

#check .vdi or .vmdk in first parametr
    if [[ $INPUT != *.vmdk ]] && [[ $INPUT != *.vdi ]] && [[ $INPUT != *.vhd
]];
    then
        echo "Incorrect first parametr, please select .vdi, .vmdk or .vhd"
        exit
    fi

#check .yml in second parametr
    if [[ $OUTPUT != *.yml ]]
    then
        echo "Incorrect second parametr, please select path to new .yml file"
        exit
    fi

}

#This function search and stores the address of a directory, where .sh was
executed
main_directory(){
SOURCE="${BASH_SOURCE[0]}"
while [ -h "$SOURCE" ]; do
  DIR="$( cd -P "$( dirname "$SOURCE" )" >/dev/null 2>&1 && pwd )"
  SOURCE="$(readlink "$SOURCE")"
  [[ $SOURCE != /* ]] && SOURCE="$DIR/$SOURCE"
done
DIR="$( cd -P "$( dirname "$SOURCE" )" >/dev/null 2>&1 && pwd )"
}

#mounting image to guestmount
mounting_image(){
    if [[ $INPUT == *.vmdk ]] || [[ $INPUT == *.vdi ]] || [[ $INPUT == *.vhd
]]; then
        if ! [ -d "$DIR/image" ]; then #create a directory if it does not
exist
        mkdir "$DIR/image";
        fi
        guestmount -a "$INPUT" -i -r "$DIR/image" #mounting image #sudo
    fi
}

#Function for the version of operation system identification
version_os(){
```

```
LINE="$(sudo cat "$DIR/image/etc/issue")"  #cat the line, where having string
version OS for DEB
for i in '\(Ubuntu 16.04\)' '\(Ubuntu 18.04\)' '\(Ubuntu 18.10\)' '\(Ubuntu
19.04\)' '\(Debian GNU/Linux 9\)' '\(Debian GNU/Linux 10\)'
do
        VERSION=`expr "$LINE" : "$i"` #regular expression for search version
        if [ "$VERSION" != "" ]; then
          break;
        fi
done


if [ "$VERSION" != "" ]; then #system family detection
SYSTEMFAMILY="DEB"
fi

#searching identificator rpm OS
if [ "$VERSION" == "" ]; then
    LINE="$(cat "$DIR/image/etc/system-release")"   #cat the line, where
having string version OS for rpm #sudo
    VERSION=`expr "$LINE" : '\(Red Hat Enterprise Linux release 8.0\)'`
#regular expression for search version
    if [ "$VERSION" == "" ]; then
        VERSION=`expr "$LINE" : '\(Red Hat Enterprise Linux release 7.7\)'`
#regular expression for search version
    fi
    if [ "$VERSION" == "" ]; then
        VERSION=`expr "$LINE" : '\(CentOS Linux release 7\)'` #regular
expression for search version
    fi
    if [ "$VERSION" == "" ]; then
        VERSION=`expr "$LINE" : '\(CentOS Linux release 8\)'` #regular
expression for search version
    fi
    if [ "$VERSION" != "" ]; then #system family detection
    SYSTEMFAMILY="RPM"
    else
    SYSTEMFAMILY="OTHER"
    fi
fi
#check OS
if [ "$VERSION" != "Ubuntu 16.04" ] && [ "$VERSION" != "Ubuntu 18.04" ] && [
"$VERSION" != "Ubuntu 18.10" ] && [ "$VERSION" != "Ubuntu 19.04" ] && [
"$VERSION" != "Debian GNU/Linux 9" ] && [ "$VERSION" != "Debian GNU/Linux 10"
] &&
[ "$VERSION" != "Red Hat Enterprise Linux release 8.0" ] && [ "$VERSION" !=
"Red Hat Enterprise Linux release 7.7" ] && [ "$VERSION" != "CentOS Linux
release 7" ] && [ "$VERSION" != "CentOS Linux release 8" ]; then
    echo "Not supported version distributive on .vdi or .vmdk"
    guestunmount "$DIR/image" #unmounting image and exit from process if
version not supported #sudo
    exit
fi

#create folders for Dockerfiles
create_dockerfile_folders(){
    HOSTNAME=`cat $DIR/image/etc/hostname`
    OUTPUT_DIR="$(dirname -- "$(readlink -f -- "$OUTPUT")")/$HOSTNAME-"$(date
+%Y-%m-%d-%H-%M-%S)""
if ! [ -d "$OUTPUT_DIR" ]; then #create a directory if it does not exist
    mkdir "$OUTPUT_DIR";
fi
```

```
if ! [ -d "$OUTPUT_DIR/configs" ]; then #create a directory if it does not
exist
    mkdir "$OUTPUT_DIR/configs";
fi
if ! [ -d "$OUTPUT_DIR/data" ]; then #create a directory if it does not exist
    mkdir "$OUTPUT_DIR/data";
fi

if ! [ -d "$DIR/docker/" ]; then
    mkdir docker/
fi

if ! [ -d "$DIR/docker/apache2" ]; then
    mkdir docker/apache2/
fi
if ! [ -d "$DIR/docker/java" ]; then
    mkdir docker/java/
fi
if ! [ -d "$DIR/docker/mariadb" ]; then
    mkdir docker/mariadb/
fi
if ! [ -d "$DIR/docker/mysql" ]; then
    mkdir docker/mysql/
fi
if ! [ -d "$DIR/docker/nginx" ]; then
    mkdir docker/nginx/
fi
if ! [ -d "$DIR/docker/phpsessionclean" ]; then
    mkdir docker/phpsessionclean/
fi
if ! [ -d "$DIR/docker/postgresql" ]; then
    mkdir docker/postgresql/
fi
if ! [ -d "$DIR/docker/redis-server" ]; then
    mkdir docker/redis-server/
fi
}

#create folders for Helms
create_helms_folders(){
if ! [ -d "$OUTPUT_DIR/helms/" ]; then
    mkdir $OUTPUT_DIR/helms/
fi

if ! [ -d "$OUTPUT_DIR/helms/apache2" ]; then
    mkdir $OUTPUT_DIR/helms/apache2/
fi
if ! [ -d "$OUTPUT_DIR/helms/java" ]; then
    mkdir $OUTPUT_DIR/helms/java/
fi
if ! [ -d "$OUTPUT_DIR/helms/mariadb" ]; then
    mkdir $OUTPUT_DIR/helms/mariadb/
fi
if ! [ -d "$OUTPUT_DIR/helms/mysql" ]; then
    mkdir $OUTPUT_DIR/helms/mysql/
fi
if ! [ -d "$OUTPUT_DIR/helms/nginx" ]; then
    mkdir $OUTPUT_DIR/helms//nginx/
fi
if ! [ -d "$DIOUTPUT_DIR/helms/phpsessionclean" ]; then
    mkdir $OUTPUT_DIR/helms/phpsessionclean/
fi
if ! [ -d "$OUTPUT_DIR/helms/postgresql" ]; then
```

```
    mkdir $OUTPUT_DIR/helms/postgresql/
fi
if ! [ -d "$OUTPUT_DIR/helms/redis-server" ]; then
    mkdir $OUTPUT_DIR/helms/redis-server/
fi
}
#variable separation $VERSION to $SYSTEMNAME and $SYSTEMVERSION
SYSTEMVERSION="`echo "$VERSION" | awk '{print $(NF)}'`"
STATUSPATH="$DIR/image/var/lib/dpkg/status"
SYSTEMNAME="`echo "$VERSION" | awk '{for(i=1;i<=NF;i++)if(i==NF)continue;else
printf("%s%s",$i,(i!=NF)?OFS:ORS)}' | cut -d " " -f1 | awk '{print
tolower($0)}'`"
}

#function to check if a package present in the VM OS
check_package(){
    CHECK="`ls "$DIR/image/lib/systemd/system/" | grep "$1.service"`" #find
name service
    if  [ -n "$CHECK" ]; then #if name service exist
    echo "$1.service" #view for debugging exiting service
    CHECK="+"
    fi
}

#generate the main .yml file containing information about the VM
yml_main(){
#generate .yml file
cat >"$OUTPUT" <<EOF
---
file:
  name: "$FILENAME"
  size: "$FILESIZE"
  type: "$FILETYPE"
system:
  family: "$SYSTEMFAMILY"
  name: "$SYSTEMNAME"
  version: "$SYSTEMVERSION"
  repositories:
  - {source_list: "`cat "$DIR/image/etc/apt/sources.list" | awk '!/^(\ )*#/
{print $0}'`"
EOF

if [[ $DIR/image/etc/apt/sources.list.d/*.list != "" ]]; then
    for var in $DIR/image/etc/apt/sources.list.d/*.list
do
cat >>"$OUTPUT" <<EOF
"`cat "$var" | awk '!/^(\ )*#/ {print $0}'`"
EOF
done
fi

cat >>"$OUTPUT" <<EOF
}
EOF
}

#added file packages.txt with installed package in disk image
system_packages(){
    cp "$DIR/image/var/lib/apt/extended_states" "$OUTPUT_DIR/packages.txt"
}
```

```
#Function is adding information in .yml file about the packages present in VM
and stores primary parameters
yml_packages(){
for var in $DIR/pkg/*.sh
do
    NAME="${var%.*}"
    NAME2=$(basename -- "$NAME")
    check_package $NAME2
    if [[ $CHECK == + ]]; then
        source $var
        cat > "$OUTPUT_DIR/helms/$NAME2/helms.yml" << EOF
---
"$IMPROVEDNAME":
  version: "$VERSION"
  mainconfig: "$MAINCONFIG"
  dbdirectory: "$DBDIRECTORY"
  port: "$PORT"
EOF
    fi
done
source ./pkg/java.sh
source ./pkg/iptables.sh
}

#run dockerfile_generate.sh
dockerfile_generate(){
    source ./dockerfile_generate.sh
}

#checking param -nfs for docker_images.py script
check_nfs_var(){
    if [ -z "$NFS" ]; then
    NFS="nonfs"
    fi
}

#push into repository
auto_tag_push(){
    check_nfs_var
    HOSTNAME=`cat $DIR/image/etc/hostname`
    python3 ./docker_images.py $HASH_ALL $OUTPUT_DIR $HOSTNAME $NFS
    #source ./docker_images.sh
}

#constant parmetrs and creating default folders
INPUT="$1" #first parametr at startup bash script
OUTPUT="$2" #second parametr at startup bash script
NFS="$3"
#FILENAME=$(basename -- "$INPUT")
FILESIZE="`ls -l --b=M  "$INPUT" | cut -d " " -f5`"
FILETYPE="VMWare"
HASH_ALL=$(sudo docker images --format "{{.ID}}")



check_parametrs #check to library guestmount and check input parametr
main_directory #search directory, where executed .sh
mounting_image #mounting image to guestmount
version_os #identificator image OS
create_dockerfile_folders #create folders for Dockerfiles
create_helms_folders #create folders for Helm
system_packages #added file packages.txt with installed package in disk image
yml_main #generate header .yml file about image info
```

```
yml_packages #added information in .yml about packages in image
auto_tag_push #push into repository

 guestunmount "$DIR/image/"
exit
```

Листинг Б.2 – Файл pre_install_hooks.sh

```
# Script for checking the necessary packages and installing them

# list of packages for installing
perm_inst(){
    sudo apt install python libguestfs-tools apache2 php mariadb-server
mariadb-client nginx postgresql redis-server apt-transport-https ca-
certificates curl gnupg gnupg-agent software-properties-common python3-pip -y
    sudo apt update
}

# installation check python modules
check_python_modules(){
    CHECK="`pip3 list | grep "python-gitlab"`"
    if ! [ -n "$CHECK" ];
    then sudo pip install python-gitlab
    fi
}

#check packages and removing
rem_old_packs(){
    CHECK="`dpkg -s docker | grep "Status"`"
    if  [ -n "$CHECK" ];
    then sudo apt remove docker -y
    fi

    CHECK="`dpkg -s docker-engine | grep "Status"`"
    if  [ -n "$CHECK" ];
    then sudo apt remove docker-engine -y
    fi

    CHECK="`dpkg -s docker.io | grep "Status"`"
    if  [ -n "$CHECK" ];
    then sudo apt remove docker.io -y
    fi

    CHECK="`dpkg -s containerd | grep "Status"`"
    if  [ -n "$CHECK" ];
    then sudo apt remove containerd -y
    fi

    CHECK="`dpkg -s runc | grep "Status"`"
    if  [ -n "$CHECK" ];
    then sudo apt remove runc -y
    fi
}

# check source list for Docker
check_sources_list(){
    CHECK="`cat /etc/apt/sources.list | grep
"https://download.docker.com/linux"`"
    if ! [ -n "$CHECK" ];
```

```
    then RETURN="0"
    else RETURN="1"
    fi
    echo $RETURN
}


#check packages and installing
inst_new_packs(){
    CHECK="`dpkg -s containerd.io | grep "Status"`"
    if ! [ -n "$CHECK" ];
    then sudo apt install containerd.io
    fi

    CHECK="`dpkg -s docker-ce | grep "Status"`"
    if ! [ -n "$CHECK" ];
    then sudo apt install docker-ce
    fi

    CHECK="`dpkg -s docker-ce-cli | grep "Status"`"
    if ! [ -n "$CHECK" ];
    then sudo apt install docker-ce-cli
    fi
}

# Changing file sources.list and adding gpg keys for Ubuntu
sources_list_edit_ubuntu(){
        RESULT=$(check_sources_list)
        if [ "$RESULT" == "0" ];
        then curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -
        sudo add-apt-repository -y "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
    fi
    inst_new_packs
}

# Changing file sources.list and adding gpg keys for Debian
sources_list_edit_debian(){
    sudo apt install software-properties-common -y
    RESULT=$(check_sources_list)
        if [ "$RESULT" == "0" ];
        then curl -fsSL https://download.docker.com/linux/debian/gpg | sudo
apt-key add -
        sudo add-apt-repository -y "deb [arch=amd64]
https://download.docker.com/linux/debian $(lsb_release -cs) stable"
    fi
    inst_new_packs
}

# Checking Linux or Debian
check_os_version(){
        LINE="$(cat "/etc/issue")"  #cat the line, where having string version
OS for DEB
    for i in '\(Ubuntu\)' '\(Debian\)'
    do
        VERSION=`expr "$LINE" : "$i"` #regular expression for search version
    done

    if [ "$VERSION" == "Ubuntu" ];
    then sources_list_edit_ubuntu
    fi
    if [ "$VERSION" == "Debian" ];
```

```
    then sources_list_edit_debian
    fi
}

perm_inst
check_python_modules
rem_old_packs
check_os_ver
```

Листинг Б.3 – Файл pkg/apache2.sh

```
# !/bin/bash

# This configuration file for a specific program - Apache


IMPROVEDNAME="Apache"

LOWERCASENAME="apache"

#VERSION_STR="`grep -REHn --colour=auto "^Package: apache2" --include='image'
| head -1 | cut -d ":" -f2`" TODO

VERSION="`$DIR/image/usr/sbin/apache2 -v | head -1 | cut -d " " -f3 | cut -d
"/" -f2`"

MAINCONFIG="$DIR/image/etc/apache2/apache2.conf"

if ! [ -d "$OUTPUT_DIR/configs/Apache" ]; then #create a directory if it does
not exist

    mkdir "$OUTPUT_DIR/configs/Apache";

fi

if ! [ -d "$OUTPUT_DIR/data/Apache" ]; then #create a directory if it does
not exist

    mkdir "$OUTPUT_DIR/data/Apache";

fi

if ! [ -d "$OUTPUT_DIR/data/Apache/www" ]; then #create a directory if it
does not exist

    mkdir "$OUTPUT_DIR/data/Apache/www";

fi
```

```
PATHSYSTEM=`grep -iRI "DocumentRoot " "$DIR/image/etc/apache2/sites-enabled/" | cut -f2 | cut -d " " -f2 | sed 's/\;\S*//'`


for i in $PATHSYSTEM; do

`cp -r "$DIR/image$i" "$OUTPUT_DIR/data/Apache/www"` #copying all file structure

done


`cp -r "$DIR/image/etc/apache2/." "$OUTPUT_DIR/configs/Apache"` #copying all file structure

#`cp -r "$DIR/image/var/www/." "$OUTPUT_DIR/data/Apache/www"` #copying all file structure

DBDIRECTORY="`grep -r -E "^<Directory" "$MAINCONFIG" | tail -1 | cut -d " " -f2 | rev | cut -c 2- | rev`"

PORT="`grep -r -E "^Listen" "$DIR/image/etc/apache2/ports.conf" | awk '{print $(NF)}'`"


cat >"$DIR/docker/$NAME2/Dockerfile" << EOF

FROM $SYSTEMNAME:$SYSTEMVERSION

ENV DEBIAN_FRONTEND=noninteractive

RUN apt update

RUN apt upgrade -y

RUN mkdir /var/run/apache2

RUN apt install apache2 -y

RUN /bin/bash -c "source /etc/apache2/envvars"

RUN apt install php -y

#ENTRYPOINT [ "apache2" ]

#COPY $OUTPUT_DIR/configs/Apache/ /etc/apache2/
```

```
EOF

docker build -t "$LOWERCASENAME" "$DIR/docker/$NAME2"


#docker run -it -v $PWD/configs/Apache:/etc/apache2 -p 80:80
bl008.setupit.org:5000/apache
```

Листинг Б.4 – Файл pkg/httpd.sh

```
# !/bin/bash


# This configuration file for a specific program - Apache for RPM


IMPROVEDNAME="Apache"

LOWERCASENAME="apache"

#VERSION_STR="`grep -REHn --colour=auto "^Package: apache2" --include='image'
| head -1 | cut -d ":" -f2`" TODO

VERSION="`$DIR/image/usr/sbin/apache2 -v | head -1 | cut -d " " -f3 | cut -d
"/" -f2`"

MAINCONFIG="$DIR/image/etc/httpd/conf/httpd.conf"

if ! [ -d "$OUTPUT_DIR/configs/Apache" ]; then #create a directory if it does
not exist

    mkdir "$OUTPUT_DIR/configs/Apache";

fi

if ! [ -d "$OUTPUT_DIR/data/Apache" ]; then #create a directory if it does
not exist

    mkdir "$OUTPUT_DIR/data/Apache";

fi

if ! [ -d "$OUTPUT_DIR/data/Apache/www" ]; then #create a directory if it
does not exist

    mkdir "$OUTPUT_DIR/data/Apache/www";

fi
```

```
`cp -r "$DIR/image/etc/httpd/." "$OUTPUT_DIR/configs/Apache"` #copying all file structure

`cp -r "$DIR/image/var/www/." "$OUTPUT_DIR/data/Apache/www"` #copying all file structure

DBDIRECTORY="`grep -r -E "^<Directory" "$MAINCONFIG" | tail -1 | cut -d " " -f2 | rev | cut -c 2- | rev`"

PORT="`grep -r -E "^Listen" "$DIR/image/etc/apache2/ports.conf" | awk '{print $(NF)}'`"


cat >"$DIR/docker/$NAME2/Dockerfile" << EOF

FROM $SYSTEMNAME:$SYSTEMVERSION

ENV DEBIAN_FRONTEND=noninteractive

RUN yum update

RUN yum upgrade -y

RUN mkdir /var/run/apache2

RUN yum install httpd -y

RUN yum install php -y

#ENTRYPOINT [ "httpd" ]

#COPY $OUTPUT_DIR/configs/Apache/ /etc/httpd/

EOF

docker build -t "$LOWERCASENAME" "$DIR/docker/$NAME2"


#docker run -it -v $PWD/configs/Apache:/etc/apache2 -p 80:80 bl008.setupit.org:5000/apache
```

Листинг Б.5 – Файл pkg/nginx.sh

```
# !/bin/bash


# This configuration file for a specific program - Nginx
```

```
IMPROVEDNAME="Nginx"

LOWERCASENAME="nginx"

VERSION="1.14.0" #`$DIR/image/usr/sbin/nginx -v | cut -d " " -f3 | cut -d "/" -f2`

MAINCONFIG="/etc/nginx"



if ! [ -d "$OUTPUT_DIR/configs/Nginx" ]; then #create a directory if it does not exist

    mkdir "$OUTPUT_DIR/configs/Nginx";

fi

if ! [ -d "$OUTPUT_DIR/configs/Nginx/share" ]; then #create a directory if it does not exist

    mkdir "$OUTPUT_DIR/configs/Nginx/share";

fi

if ! [ -d "$OUTPUT_DIR/data/Nginx/" ]; then #create a directory if it does not exist

    mkdir "$OUTPUT_DIR/data/Nginx/";

fi

if ! [ -d "$OUTPUT_DIR/data/Nginx/lib" ]; then #create a directory if it does not exist

    mkdir "$OUTPUT_DIR/data/Nginx/lib";

fi

if ! [ -d "$OUTPUT_DIR/data/Nginx/www" ]; then #create a directory if it does not exist

    mkdir "$OUTPUT_DIR/data/Nginx/www";

fi
```

```
PATHSYSTEM=`grep -iRI "root " "$DIR/image/etc/nginx/sites-enabled/" | cut -f2
| cut -d " " -f2 | sed 's/\;\S*//'`


for i in $PATHSYSTEM; do

`cp -r "$DIR/image$i" "$OUTPUT_DIR/data/Nginx/www"` #copying all file
structure

done


`cp -r "$DIR/image/etc/nginx/." "$OUTPUT_DIR/configs/Nginx"` #copying all
file structure

`cp -r "$DIR/image/usr/share/nginx/." "$OUTPUT_DIR/configs/Nginx/share"`
#copying all file structure

`cp -r "$DIR/image/usr/lib/nginx/." "$OUTPUT_DIR/data/Nginx/lib"` #copying
all file structure

DBDIRECTORY="—"

PORT="`grep -r -E "listen" "$DIR/image/etc/nginx/sites-available/default" |
head -1 | cut -d " " -f2`"


cat >"$DIR/docker/$NAME2/Dockerfile" << EOF

FROM $SYSTEMNAME:$SYSTEMVERSION

ENV DEBIAN_FRONTEND=noninteractive

#RUN echo "deb http://archive.ubuntu.com/ubuntu/ bionic main restricted
universe multiverse" >> /etc/apt/sources.list

RUN apt update

RUN apt upgrade -y

RUN apt install nginx -y

ENTRYPOINT nginx -g 'daemon off;'

#COPY $OUTPUT_DIR/configs/Nginx/ /etc/nginx/

EOF
```

```
docker build -t "$LOWERCASENAME" "$DIR/docker/$NAME2"


#docker run -it -v $PWD/configs/Nginx/:/etc/nginx -v
$PWD/configs/Nginx/share/:/usr/share/nginx -v
$PWD/configs/Nginx/lib/:/usr/lib/nginx -p 80:80 nginx:latest
```

Листинг Б.6 – Файл pkg/postgresql.sh

```
# !/bin/bash


# This configuration file for a specific program - PostgreSQL


IMPROVEDNAME="PostgreSQL"

LOWERCASENAME="postgresql"

VERSION="`ls "$DIR/image/etc/postgresql/"`"

MAINCONFIG="$DIR/image/etc/postgresql/"$VERSION"/main/postgresql.conf"

if ! [ -d "$OUTPUT_DIR/configs/PostgreSQL" ]; then #create a directory if it
does not exist

    mkdir "$OUTPUT_DIR/configs/PostgreSQL";

fi

`cp $(echo "$MAINCONFIG" | awk '{print $(NF)}')
"$OUTPUT_DIR/configs/PostgreSQL/postgresql.conf"` #copying .conf file

DBDIRECTORY="`grep -r data_directory "$MAINCONFIG" | awk '{print $3}'`"
#expression for find database directory in postgresql.conf

DBDIRECTORYCP=${DBDIRECTORY//"'"/}

PORT="`grep -r -E "^port" "$MAINCONFIG" | awk '{print $3}'`" #expression for
find port in postgresql.conf


if ! [ -d "$OUTPUT_DIR/data/PostgreSQL" ]; then #create a directory if it does
not exist
```

```
    mkdir "$OUTPUT_DIR/data/PostgreSQL";
fi
if ! [ -d "$OUTPUT_DIR/data/PostgreSQL/db" ]; then #create a directory if it does not exist
    mkdir "$OUTPUT_DIR/data/PostgreSQL/db";
fi
`cp -r "$DIR/image$DBDIRECTORYCP/." "$OUTPUT_DIR/data/PostgreSQL/db"` #copying all file structure



cat >"$DIR/docker/$NAME2/Dockerfile" << EOF
FROM $SYSTEMNAME:$SYSTEMVERSION
ENV DEBIAN_FRONTEND=noninteractive
#RUN echo "deb http://archive.ubuntu.com/ubuntu/ bionic main restricted universe multiverse" >> /etc/apt/sources.list
RUN apt update
RUN apt upgrade -y
RUN apt install postgresql -y
#COPY $OUTPUT_DIR/configs/PostgreSQL/postgresql.conf /etc/postgresql/"$VERSION"/main/
EOF
docker build -t "$LOWERCASENAME" "$DIR/docker/$NAME2"
```

Листинг Б.7 – Файл docker_images.py

```
import os
import sys
import yaml
```

```
from parsermodules.mainparse import mainparser


'''
This script doing a tagging and pushing to private docker repository.
Also sends parameters to next scripts
'''


'''This block of code define connection strings for connection to your remote
private repository'''


all_configs = {}
all_constants = {}
# ==USER DATA==
login = ''
password = ''
repo = ''
mode = ''  # corporate or dockerhub


# ==GIT CREDENTIALS FOR INIT GIT REPO==
git_login = ''
git_password = ''
git_address = ''  # mustn't include an https:// prefix
nfs_default_path = ''
# ==================


names = []
```

```
# function that execute pushing our generated images into remote private repository

def local_repo_push():

    for name in names:

        os.system('sudo docker push ' + repo + '/' + name)




# function that execute tagging the generated images by repository name in local registry

def docker_retag():

    for name in names:

        os.system('sudo docker tag ' + name + ' ' + repo + '/' + name)

        os.system('sudo docker rmi ' + name)

    local_repo_push()




def docker_login():

    login_credit = repo

    if mode == 'dockerhub':

        login_credit = ' '

    os.system('sudo docker login -u ' + login + ' -p ' + password + ' ' + login_credit)

    docker_retag()




# check nfs value

def nfs_checking(input_nfs):
```

```
    if input_nfs == 'nonfs':
        return nfs_default_path
    return input_nfs



# parsing of yaml config file
def config_parse():
    global all_configs
    all_configs = {}
    try:
        with open(os.getcwd() + '/config.yaml', 'r') as stream:
            try:
                all_configs = yaml.safe_load(stream)
            except yaml.YAMLError as exc:
                print(exc)
    except FileNotFoundError:
        print(all_constants['errors']['err_config_parse'])
        raise SystemExit



# parsing of yaml config file into dictionary
def constants_parse():
    global all_constants
    all_constants = {}
    try:
        with open(os.getcwd() + '/constants.yaml', 'r') as stream:
            try:
```

```
                all_constants = yaml.safe_load(stream)
            except yaml.YAMLError as exc:
                print(exc)
    except FileNotFoundError as fof:
        print(fof)
        raise SystemExit
    if all_constants['gitRepoName'] == '':
        print(all_constants['errors']['err_gitRepoName'])
        raise SystemExit



# setting values from config file
def setting_values():
    global login, password, repo, mode, git_login, git_password, git_address,
nfs_default_path
    login = all_configs['docker']['login']
    password = all_configs['docker']['password']
    repo = all_configs['docker']['repo']
    mode = all_configs['docker']['mode']
    git_login = all_configs['git']['git_login']
    git_password = all_configs['git']['git_password']
    git_address = all_configs['git']['git_address']  # mustn't include an
https:// prefix
    nfs_default_path = all_configs['git']['nfs_default_path']



if __name__ == "__main__":
```

```
    constants_parse()

    config_parse()

    setting_values()


    length = len(sys.argv)

    nfs = nfs_checking(sys.argv[length - 1])


    if length == 4:

        temp = os.popen('sudo docker images --format
"{{.Repository}}"').read()

        names = temp.split()

    else:

        temp = os.popen('sudo docker images --filter "since=' + sys.argv[1] +
'" --format "{{.Repository}}"').read()

        names = temp.split()

    docker_login()

    mainparser(repo, sys.argv[length-3], nfs, git_login, git_password,
git_address, all_constants,

               all_configs['valuesFileName'], all_configs['helmsFileName'],
all_configs['git']['proto'],

               all_configs['git']['branch'])


    '''

    # ===test_values===


    nfs = nfs_checking('nonfs')

    mainparser(repo=repo, output_dir=os.getcwd(), nfs=nfs,
git_login=git_login,
```

```
                git_password=git_password, git_address=git_address,
all_consts=all_constants,

                values_name=all_configs['valuesFileName'],
helms_name=all_configs['helmsFileName'],

                protocol=all_configs['git']['proto'],
cur_branch=all_configs['git']['branch'])
```