МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2020 г.

ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ ПЛАТФОРМ УПРАВЛЕНИЯ ВЫЧИСЛИТЕЛЬНЫМИ СЕРВИСАМИ ПРИ ОРГАНИЗАЦИИ ТУМАННЫХ ВЫЧИСЛЕНИЙ

Руководитель работы,
к.ф-м.н., доцент каф. ЭВМ
_____Г.И. Радченко
«___»_____2020 г.

Автор работы,
студент группы КЭ-405
_____А.А. Асташов
«___»_____2020 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2020 г.

Челябинск 2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2020 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Асташов Андрей Алексеевич
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Исследование эффективности платформ управления вычислительными сервисами при организации туманных вычислений» утверждена приказом по университету от № 627 от 24.04.2020

2. **Срок сдачи студентом законченной работы:** 1 июня 2020 г.

3. **Исходные данные к работе:** статьи, книги, техническое задание.
   - Ли, П. Архитектура интернета вещей / П. Ли; пер. с анг. М. А. Райтмана. – М.: ДМК Пресс, 2019. – 454 с.
   - Электронная документация Docker Swarm. [Электронный ресурс] URL: https://docs.docker.com/engine/reference/commandline/swarm/
   - Электронная документация Docker. [Электронный ресурс] URL: https://docs.docker.com

- Yousefpour, A. All one needs to know about fog computing and related edge computing paradigms: A complete survey / A. Yousefpour , C. Fung , T. Nguyen , K. Kadiyala , F. Jalali , A. Niakanlahiji , J. Kong , J. P. Jue // Journal of Systems Architecture. – 2019. – P. 1–44.
- Гордеев, А.В. Сравнительное тестирование контейнерной и гипервизорной виртуализации / А.В. Гордеев, Д.В. Горелик // СПбГУАП, Информационно-управляющие системы №2, 2018. – С. 60–66.
- Баранов, А.В. Использование контейнерной виртуализации в организации высокопроизводительных вычислений / А.В. Баранов, Д.С. Николаев // Программные системы: Теория и приложения. №1(28), 2016. – С. 117-134.

4. **Перечень подлежащих разработке вопросов:**
   - произвести обзор литературы, необходимой для проведения исследования;
   - выполнить обзор существующих решений контейнерной виртуализации;
   - определить ключевые требования и критерии исследования;
   - спроектировать, реализовать решения для проведения исследования;
   - выполнить исследование.

5. **Дата выдачи задания:** 1 декабря 2019 г.

Руководитель работы _____ Г.И. Радченко

Студент _____ А.А. Асташов

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2020	
Разработка модели, проектирование	01.04.2020	
Реализация системы	01.05.2020	
Тестирование, отладка, эксперименты	15.05.2020	
Компоновка текста работы и сдача на нормоконтроль	24.05.2020	
Подготовка презентации и доклада	30.05.2020	

Руководитель работы _____ Г.И. Радченко

Студент _____ А.А. Асташов

Аннотация

А.А. Асташов. Исследование эффективности платформ управления вычислительными сервисами при организации туманных вычислений. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2020, 46 с., 39 ил., библиогр. список – 15 наим.

В рамках выпускной квалификационной работы производится исследование эффективности платформ контейнерной оркестрации при организации туманных вычислений.

Организуется развертывание Docker-контейнеров. Для создания кластера используется Docker Swarm. Решаются задачи измерения следующих параметров: время развертывания одного контейнера, время развертывания группы контейнеров, время отклика для задачи горизонтального масштабирования, время задержки передачи данных. Производится анализ полученных результатов испытаний.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 8
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 11
1.1 Облачные и туманные вычисления 11
1.2 Виртуализация и контейнеризация 12
1.3 Инструменты контейнеризации 15
1.4 Контейнерная оркестрация 16
1.5 Выводы 17
2 ПРОЕКТИРОВАНИЕ 18
2.1 Общие сведения 18
2.2 Определение требований к исследованию 18
2.2.1 Проведение испытания «Один контейнер» 19
2.2.2 Проведение испытания «Группа контейнеров» 20
2.2.3 Проведение испытания «Горизонтальное масштабирование» 20
2.2.4 Проведение испытания «Latency» 21
2.3 Система проведения испытания 22
2.4 Метод оценки времени выполнения испытания 23
3 РЕАЛИЗАЦИЯ 25
3.1 Разработка утилиты автоматизации испытаний 25
3.2 Настройка хостов в Amazon Web Service 26
3.3 Запуск кластера Docker Swarm 28
3.4 Реализация слушателя 30
3.5 Реализация клиента 30
4 ПРОВЕДЕНИЕ ИСПЫТАНИЙ 32
4.1 Время развертывания одного контейнера в кластере 32
4.2 Время развертывания группы контейнеров в кластере 33

4.3 Время горизонтального масштабирования существующих контейнеров 35
4.4 Временные задержки на передачу сообщений между сервисами в кластере 37
4.5 Вывод 39
5. ЗАКЛЮЧЕНИЕ 43
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 44
ПРИЛОЖЕНИЕ 46

ВВЕДЕНИЕ

Актуальность исследовательской темы

На сегодняшний день, когда технологии интернета вещей и предоставление облачных вычислений как сервисов услуг уверенно находят свое применение, развиваются новые вычислительные концепции, в частности технологии туманных вычислений. Данный вид вычислений до сих пор является новой технологией, которая только начинает набирать свою популярность.

Парадигма облачных вычислений способствовала развитию предоставления вычислительных сервисов и инфраструктур как услуг, что позволило сэкономить средства на создании и обслуживании собственной вычислительной инфраструктуры. В этой среде упрощено и автоматизировано масштабирование приложений для удовлетворения потребностей в условиях высокой нагрузки. Виртуализация является ключевой технологией, обеспечивающей данные возможности. В настоящее время контейнеризация стала популярной альтернативой виртуальным машинам и получила широкое применение, в результате чего инструменты оркестрации стали неотъемлемой частью облачных вычислений. Несмотря на успешное применение технологии контейнеризации, до сих пор облачные вычисления не обеспечивают должного соответствия критериям технологий интернета вещей. Управление службами, развернутыми в туманной вычислительной среде, представляет собой сложную задачу, а инструменты контейнеризации и оркестрации реализуют ее беспроблемное внедрение и использование.

Таким образом, исследование эффективности платформ контейнерной оркестрации при организации туманных вычислений является актуальной задачей развития концепции туманных вычислений.

Цель и задачи исследования

В ходе проведения исследования необходимо выполнить следующие задачи:

1) произвести подбор литературы, научных публикаций и интернет статей, необходимых для проведения исследования;
2) выполнить обзор платформ контейнерной оркестрации;
3) определить ключевые требования и критерии проведения исследования;
4) спроектировать и реализовать утилиту автоматического проведения испытаний;
5) выполнить исследование эффективности платформ контейнерной оркестрации при организации туманных вычислений;
6) проанализировать полученные результаты и сделать сопутствующие выводы.

Структура и объем работы

Работа состоит из введения, 4 разделов, заключения, библиографического списка, приложения.

Работа составляет 46 страниц, в библиографическом списке указано 15 источников, объем приложения – 1 страница.

В первой главе производится обзор научных публикаций по тематике исследования и анализ предметной области, приведен обзор существующих наиболее популярных платформ.

Во второй главе раскрывается проектирование утилиты автоматического проведения испытаний, ее составляющие, приведены поясняющие диаграммы. Также производится определение требований к исследованиям, описание проведения испытаний и метода оценки времени выполнения испытания.

Третья глава описывает реализацию утилиты автоматического проведения испытаний, настройку и запуск кластера в Amazon Web Services, реализацию контейнеризованных программ-клиентов, реализацию программы-слушателя.

В четвертой главе приводятся описание и результаты проведения испытаний.

В заключении описывается анализ полученных результатов испытаний, делается итоговый вывод.

В приложении располагается листинг программного кода утилиты автоматического проведения испытаний.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Облачные и туманные вычисления

На сегодняшний день количество устройств интернета вещей растет каждый день и, в связи с этим, появляется необходимость в удобной системе, способной обрабатывать, хранить и передавать данные. Сейчас для этих целей используют облачные сервисы [1].

За последнее десятилетие облачный подход создания приложений внес значительные изменения в IT-индустрию [2]. Облачные вычисления освобождают предприятие и конечного потребителя от спецификации многих деталей, что становится проблемой для приложений и устройств, чувствительных ко времени ожидания и требуют нахождения узлов в непосредственной близости друг от друга [3].

Узкое место облачного подхода – полоса связи между периферией и облаком. Чтобы разгрузить этот канал, необходимо снизить количество данных, передаваемых в облако. Также, для поддержки чувствительной ко времени machine-to-machine коммуникации надлежит снизить временную задержку транзакций между конечными устройствами и аналитическим аппаратом. Для этого было предложено переместить аналитический аппарат из облака к периферийным устройствам. В этом и есть суть концепции туманных вычислений [2].

Облачные и туманные вычисления плодотворно взаимодействуют, позволяя создать новое поколение приложений и услуг на их границе, особенно в случае, когда дело доходит до управления данными и аналитики [3].

Облачные сервисы способны закрыть множество запросов интернета вещей. Например, обеспечить мониторинг служб, быструю обработку любых объемов данных, генерируемых устройствами, а также их визуализации.

Туманные же вычисления эффективнее при решении задачи реального времени. Они обеспечивают быстрый отклик на запросы и минимальную задержку при обработке данных. То есть туманные вычисления являются дополнением облака, расширяющие его возможности [1].

Туманные вычисления – это высоко-виртуализированная платформа, предоставляющая вычислительные, запоминающие и сетевые сервисы между конечными устройствами и центрами данных облачных вычислений, и которая обычно, но не всегда, располагается на границах сети.

Инфраструктура туманных и облачных вычислений подразумевает использование и размещение каждого приложения по отдельности, другими словами, изолированно друг от друга. Чтобы достичь этого, используется виртуализация, виртуальные машины и технологии контейнеризации.

Подобно облачным сервисам, инфраструктура туманных вычислений требует автоматического управления множеством контейнеризированных приложений. Для этого полезно использовать платформы оркестрации контейнеров [4**].**

1.2 Виртуализация и контейнеризация

Виртуализация и облачные вычисления – в равной степени инновационные технологии. Средства виртуализации – программное обеспечение, которое помогает сделать вычислительные среды независимыми от физической инфраструктуры, а облачные вычисления – службы, которые предоставляют общие вычислительные ресурсы по требованию [5].

Сегодня, благодаря облачной революции на первый план вышли такие требования к центрам обработки данных, как эластичность, масштабируемость и высокая вычислительная плотность, контейнеры стали предметом

повышенного интереса – прежде всего потому, что они как нельзя лучше подходят для решения названных задач.

В самых общих чертах, виртуализация – способ запуска одной операционной системы поверх другой [6].

Гипервизор или аппаратная виртуализация – программа или аппаратная схема, обеспечивающая или позволяющая одновременное параллельное выполнение нескольких операционных систем на одном и том же хост-компьютере. Он обеспечивает изоляцию операционных систем друг от друга. Виртуализация на базе гипервизора работает следующим образом: в операционной системе хоста эмулируется аппаратное обеспечение, поверх которого запускаются гостевые операционные системы [7]. На рисунке 1 можем видеть схему виртуализации на базе гипервизора.

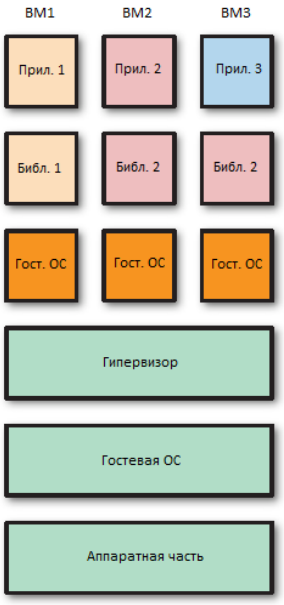

Рисунок 1 – Виртуализация на базе гипервизора

В отличие от аппаратной виртуализации, когда эмулируется аппаратное окружение и может быть запущено множество гостевых операционных систем, в контейнере запускается экземпляр операционной системы только с тем же ядром, что и у исходной (хостовой) операционной системы. Таким образом, все

контейнеры узла используют общее ядро. При этом отсутствуют дополнительные накладные расходы ресурсов на эмуляцию виртуального оборудования и запуск полноценного экземпляра операционной системы, что характерно аппаратной виртуализации. На рисунке 2 можем видеть схему контейнерной виртуализации.

Рисунок 2 – Контейнерная виртуализация

Таким образом, контейнеры позволяют уместить гораздо больше приложений на одном физическом сервере, чем любая виртуальная машина, которая занимает гораздо больше системных ресурсов. В отличие от виртуальной машины, где эмулируется операционная система и необходимое виртуальное оборудование, в контейнере размещается только приложение и необходимый минимум системных библиотек. Благодаря этому можно запустить в 2-3 раза больше приложений на одном сервере. Также контейнеризация позволяет создавать портативное и целостное окружение для разработки, тестирования и последующего развертывания [8].

1.3 Инструменты контейнеризации

Когда мы говорим о контейнерах в современных информационных системах, прежде всего мы подразумеваем Docker – open-source технологию, благодаря своей популярности ставшую в информационных технологиях синонимом слова «контейнер» [9].

В основу Docker была заложена существующая технология Linux-контейнеров с разнообразными обертками и расширениями – в основном использующими переносимые образы и удобный для пользователя интерфейс – для создания полностью готового к применению решения, обеспечивающего создание и распространение контейнеров. Платформа Docker состоит из двух отдельных компонентов: Docker Engine, механизма, отвечающего за создание и функционирование контейнеров, и Docker Hub, облачного сервиса для распространения контейнеров. Механизм Docker Engine предоставляет эффективный и удобный интерфейс для запуска контейнеров. До этого для запуска контейнеров, использующих такую технологию, как, например, LXC, требовались изрядный запас специальных знаний в этой области и большой объем ручной работы. Docker Hub предоставляет огромное количество образов контейнеров с открытым доступом для загрузки, позволяя пользователям быстро начать работу с ними и избежать рутинной работы, ранее уже проделанной другими людьми. Несколько позже были разработаны инструментальные средства для Docker: Swarm – менеджер кластеров, Kinematic – графический пользовательский интерфейс для работы с контейнерами и Machine – утилита командной строки для поддержки работы Docker-хостов [10].

1.4 Контейнерная оркестрация

Контейнерная оркестрация – это организация совместной работы множества сервисов в рамках одного приложения. На рисунке 3 можем видеть схему контейнеров, объединенных в кластер.

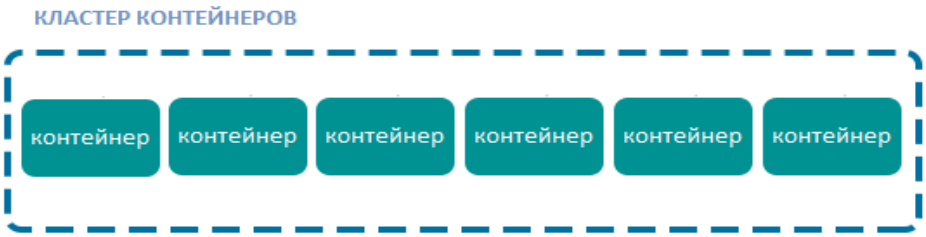

Рисунок 3 – Кластер контейнеров

На рынке программного обеспечения доступно множество решений для оркестрации контейнеров, которые могут управлять полным жизненным циклом контейнеров или группой контейнеров, масштабируя их вертикально или горизонтально, перемещая, самостоятельно восстанавливая, выполняя оркестрацию хранения и другие функции [11].

Одними из наиболее популярных решений для обеспечения оркестрации контейнеров на сегодняшний день являются Kubernetes и Docker Swarm [12].

Docker Swarm – это режим Docker, который обеспечивает возможность объединения физических и виртуальных машин в один виртуальный хост. Docker в режиме Swarm предоставляет REST API интерфейс, совместимый с Docker API. Все инструменты, которые работают с API Docker, смогут работать с Docker Swarm, не подозревая о том, что за ним стоит кластер, состоящий из нескольких хостов.

Docker Swarm состоит из управляющих (manager) и рабочих (worker) нод, а так же обеспечивает автоматическое масштабирование, самоорганизацию и безопасность коммуникации.

Kubernetes - инструмент оркестрации с открытым исходным кодом для автоматизации развертывания, масштабирования и управления приложениями на основе Docker-контейнеров. Можно запускать Kubernetes в локальной или публичной облачной инфраструктуре. Kubernetes устанавливает кластер, состоящий из Kubernetes-master, Kubernetes-minions (рабочие узлы) и Pod (один или группа контейнеров для запуска одного приложения). Kubernetes-master наблюдает за одним или несколькими миньонами и управляет приложениями.

Kubernetes управляет и запускает контейнеры на большом количестве хостов, а так же обеспечивает совместное размещение и репликацию большого количества контейнеризированных контейнеров. Kubernetes обеспечивает высокую надежность контейнеров, эффективно использует доступные ресурсы, координирует развертывание и реализует взаимодействие контейнеров между собой.

1.5 Выводы

Из представленной информации можно сделать вывод о том, что развитие технологии облачных вычислений поспособствовало развитию технологии контейнеризации и появлению концепции туманных вычислений.

Туманные вычисления появились из-за недостаточного соответствия облачных вычислений требованиям сферы интернета вещей.

Для организации инфраструктуры туманных вычислений применяются те же технологии контейнеризации, что и для облачных вычислений с учетом того, что такие вычислительные узлы должны располагаться в непосредственной близости к конечным устройствам и сочетать в себе набор необходимых сервисов.

Для управления, масштабирования и контроля контейнеризированных сервисов применяются платформы оркестрации контейнеров.

2 ПРОЕКТИРОВАНИЕ

2.1 Общие сведения

Целью данной работы является исследование эффективности платформ оркестрации контейнеризированных вычислительных систем при организации туманных вычислений.

В данном исследовании будут проведены следующие испытания для платформы оркестрации контейнеризованных приложений Docker Swarm:

- измерение времени развертывания одного контейнера в кластере;
- измерение времени развертывания группы контейнеров в кластере;
- измерение времени горизонтального масштабирования существующих контейнеров;
- измерение временных задержек между сервисами в кластере.

2.2 Определение требований к исследованию

Для достоверного проведения исследования для разных инструментов контейнерной оркестрации необходимо создать идентичные условия выполнения тестирований и провести ряд повторений каждого испытания для достижения достоверных результатов. Для этого воспользуемся предоставляемыми Amazon Web Services виртуальными машинами. Перечислим значимые характеристики:

- количество виртуальных машин: 3;
- предустановленная операционная система: Linux Ubuntu Server 16.04 LTS x64;
- количество процессорных ядер каждой виртуальной машины: 2;
- объем оперативной памяти каждой виртуальной машины: 4 Гб.

Для удобства и информативности введем сокращенные наименования для каждого испытания и представим в табличном виде (таблица 1).

Таблица 1 – Наименования испытаний.

Наименование испытания	**Сокращенное наименование испытания**
Измерение времени развертывания одного контейнера	Один контейнер
Измерение времени развертывания группы контейнеров	Группа контейнеров
Измерение времени горизонтального масштабирования существующих контейнеров	Горизонтальное масштабирование
Измерение временных задержек между сервисами в кластере	Latency

2.2.1 Проведение испытания «Один контейнер»

Проведение испытания «Один контейнер» должно обеспечивать:

- развертывание одного контейнера в кластере;
- измерение времени выполнения развертывания.

Испытание «Один контейнер» должно быть реализовано следующим образом:

- запуск слушателя;
- фиксация времени начала испытания;
- выполнение команды развертывания одного контейнера;
- фиксация времени получения отклика от контейнера;
- демонстрация времени начала испытания и получения отклика от контейнера;
- расчёт времени развертывания контейнера.

2.2.2 Проведение испытания «Группа контейнеров»

Проведение испытания «Группа контейнеров» должно обеспечивать:

- развертывание группы контейнеров в кластере;
- измерение времени выполнения развертывания.

Испытание «Группа контейнеров» должно быть реализовано следующим образом:

- запуск слушателя;
- фиксация времени начала испытания;
- выполнение команды развертывания группы контейнеров с указанием количества реплик;
- фиксация времени получения отклика от каждого контейнера;
- демонстрация времени начала испытания и получения отклика от каждого контейнера;
- расчёт времени развертывания последнего контейнера.

2.2.3 Проведение испытания «Горизонтальное масштабирование»

Проведение испытания «Горизонтальное масштабирование» должно обеспечивать:

- увеличение числа реплик ранее развернутого сервиса (с 1 до 16, с 16 до 64);
- измерение времени выполнения масштабирования.

Испытание «Горизонтальное масштабирование» должно быть реализовано следующим образом:

- запуск слушателя;
- фиксация времени начала испытания;
- выполнение команды масштабирования с указанием количества реплик;

– фиксация времени получения отклика от вновь-масштабируемых контейнеров;
– демонстрация времени начала испытания;
– демонстрация времени получения ответа от вновь-созданных контейнеров;
– расчёт времени развертывания последнего контейнера.

2.2.4 Проведение испытания «Latency»

Проведение испытания «Latency» подразумевает:

– настройку внутренней сети кластера;
– подключение двух сервисов к внутренней сети;
– измерение временных задержек при обмене пакетами данных между двумя контейнерами разных сервисов;
– сбор результатов.

Испытание «Latency» должно быть реализовано следующим образом:

– создаем собственную ingress сеть;
– создаем и подключаем к созданной сети 2 сервиса;
– запускаем терминал в первом контейнере;
– устанавливаем утилиту ping;
– выполняем 10 запросов по IP-адресу сети ingress второго контейнера;
– производим сбор данных пункта time утилиты ping.

В качестве развертываемого контейнера используется контейнер с клиентом, который отправляет на слушателя сообщение о его развертывании.

В качестве слушателя используется сервер, принимающий TCP-соединения, который фиксирует время начала проведения испытания и время каждого принятого сообщения

Повторение каждого испытания будет повторяться 10 раз.

Чтобы исключить человеческий фактор при проведении каждого испытания, должна быть реализована утилита автоматического проведения испытаний.

При использовании утилиты автоматического проведения испытаний, время измерения испытания должно быть показано в конце каждого испытания.

Результаты испытаний измеряются в миллисекундах.

После проведения измерений должен быть произведен сравнительный анализ полученных результатов и сделаны сопутствующие выводы.

2.3 Система проведения испытания

Для достоверности результатов проведения испытания необходимо разработать утилиту автоматического проведения испытаний, которая должна обеспечивать следующий функционал:

- возможность настройки испытания;
- выбор и запуск испытания;
- мониторинг текущего испытания;
- демонстрация результатов испытания.

Для описания вариантов использования утилиты разработана диаграмма вариантов использования (рисунок 4) – диаграмма, отражающая отношения между акторами и прецедентами.

Рисунок 4 – Диаграмма прецедентов утилиты автоматического проведения испытаний

2.4 Метод оценки времени выполнения испытания

Для того, чтобы измерить время проведения испытаний был выбран принцип обратного отклика каждого развернутого контейнера [13], который заключается в следующем:

- запускается слушатель (сервер), который принимает сообщения от контейнеров (клиентов);
- слушатель фиксирует время начала испытания;
- слушатель принимает все сообщения, посылаемые контейнерами при развертывании, и демонстрирует время отклика.

Чтобы получить время испытания, необходимо из времени последнего отклика вычесть время начала испытания.

Наглядно принцип обратного отклика каждого контейнера можно увидеть на рисунке 5.

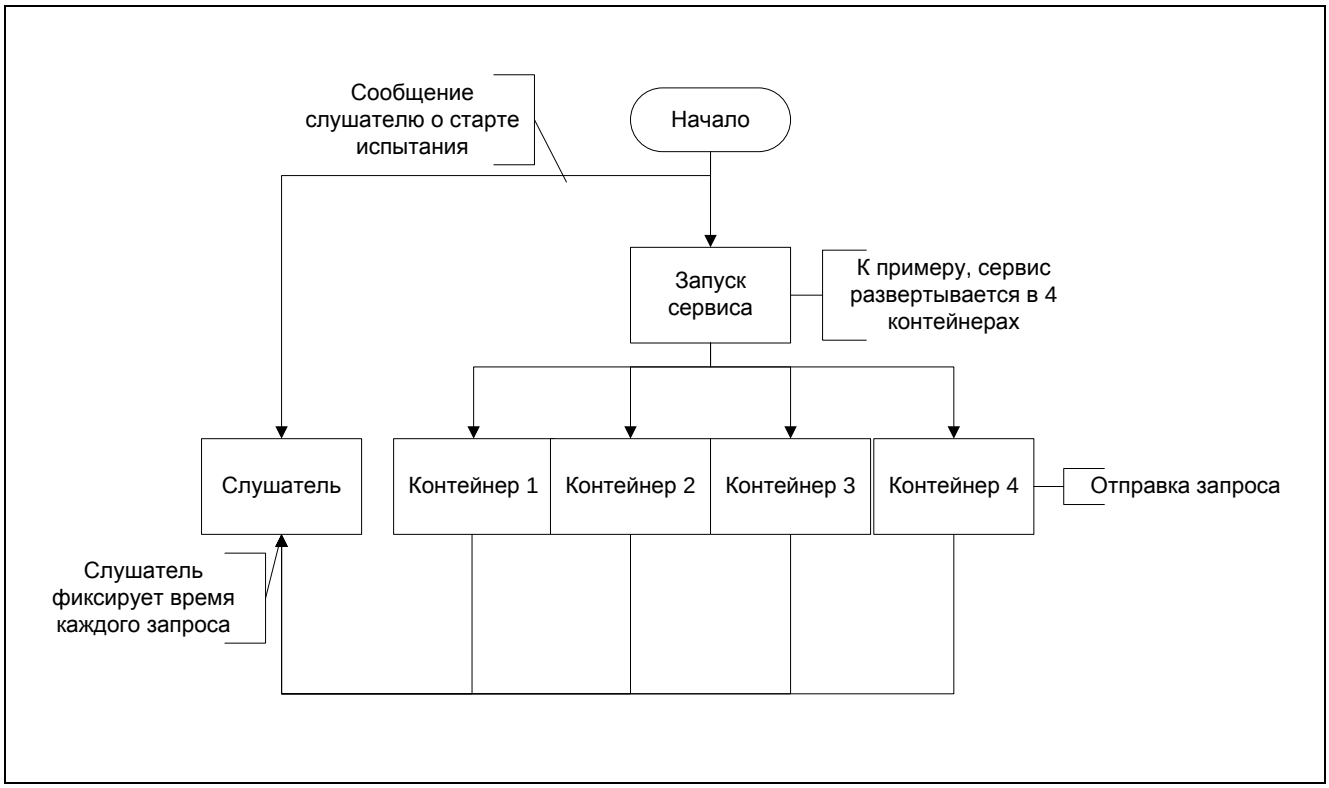

Рисунок 5 – Принцип обратного отклика каждого контейнера

3 РЕАЛИЗАЦИЯ

3.1 Разработка утилиты автоматизации испытаний

Для автоматизации проведения испытаний была разработана утилита автоматического развертывания и масштабирования контейнеров на языке bash.

Блок-схема утилиты продемонстрирована на рисунках 6–7.

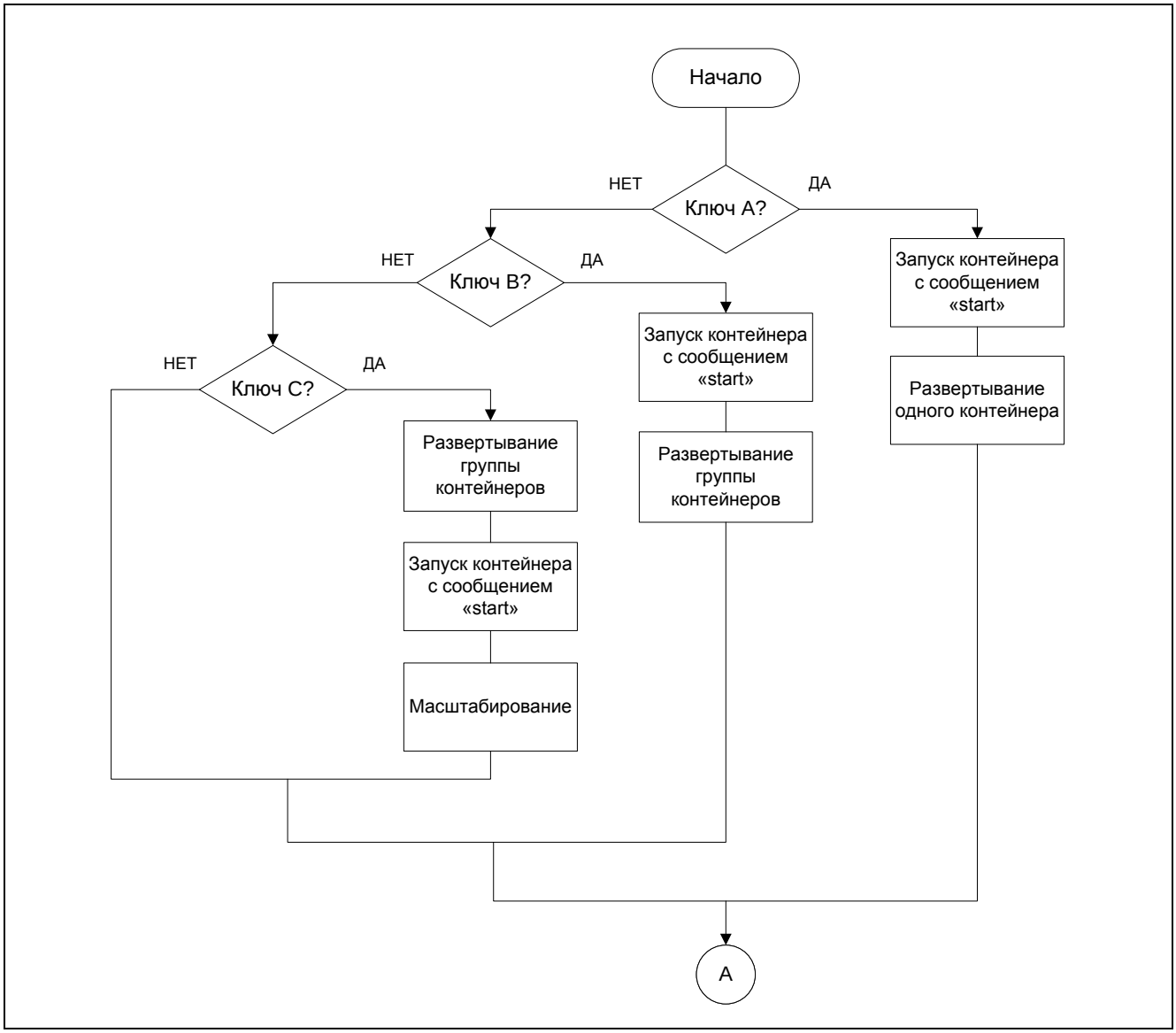

Рисунок 6 – Блок-схема утилиты (фрагмент 1)

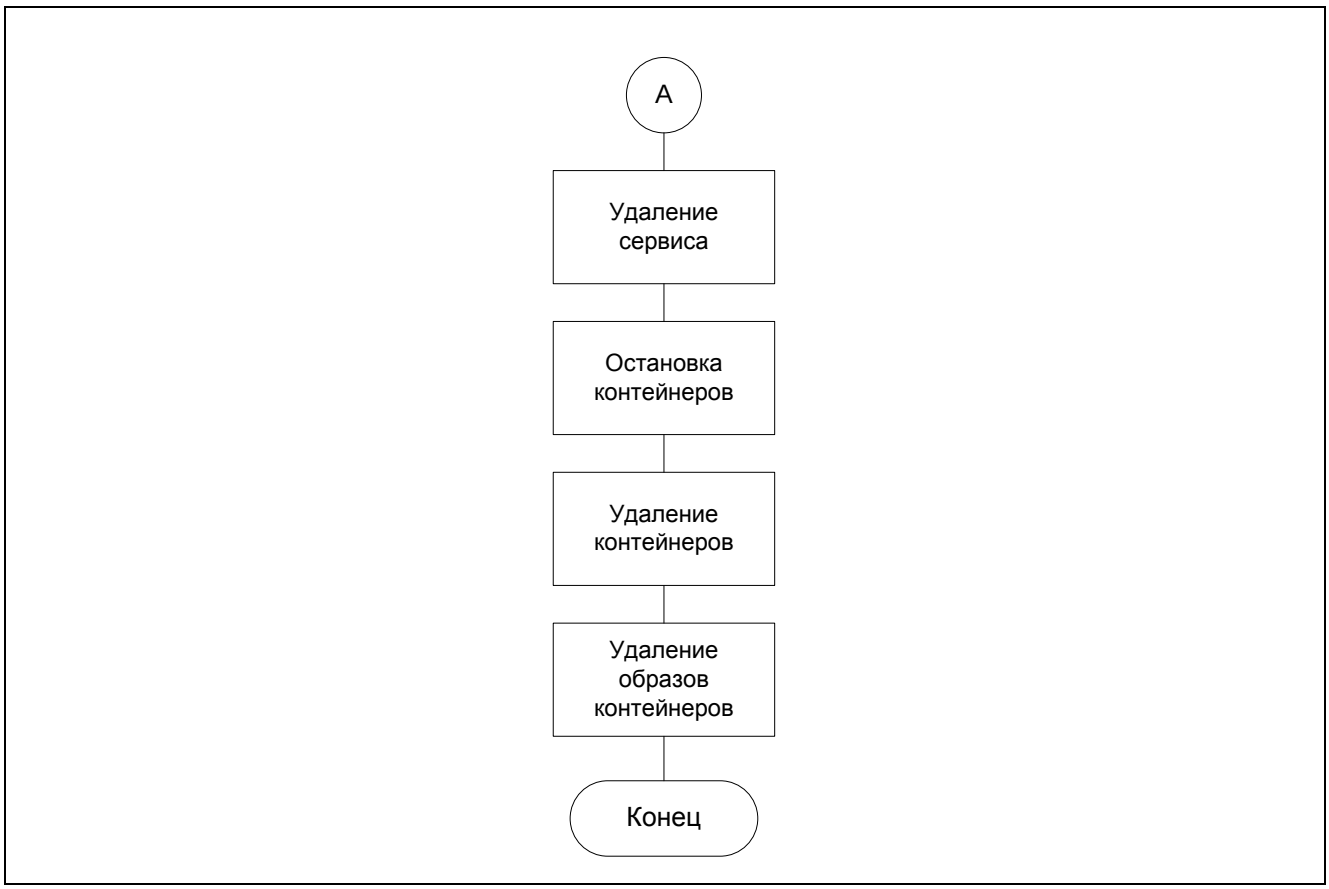

Рисунок 7 – Блок-схема утилиты (фрагмент 2)

Исходный код утилиты автоматического проведения испытаний расположен в разделе «Приложение» – листинг 1.

3.2 Настройка хостов в Amazon Web Service

Для проведения исследования был выбран Amazon Web Service в качестве сервиса, предлагающего услуги использования вычислительных мощностей.

Для проведения испытаний нам понадобятся виртуальные машины уровня t2.medium (vCPU: 2, RAM: 4 Гб) в количестве 3 экземпляров с предустановленной операционной системой Ubuntu Server 16.04 LTS.

Создадим и проинициализируем виртуальные машины (рисунок 8).

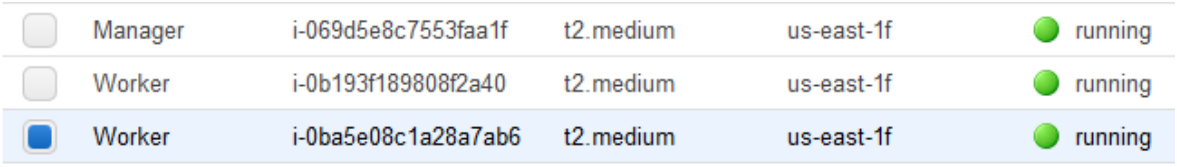

Рисунок 8 – Виртуальные машины в AWS

Так как виртуальные машины не содержат необходимого предустановленного программного обеспечения (Docker), установим на каждой виртуальной машине его вручную.

Для установки Docker на виртуальные машины обратимся к официальной документации за инструкцией по установке.

Выполним поочередно следующие команды для каждой из виртуальных машин (рисунки 9–10):

```
$ sudo apt-get update

$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

Рисунок 9 – Команды установки Docker (фрагмент 1)

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

$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Рисунок 10 – Команды установки Docker (фрагмент 2)

Проверим состояние установленного программного обеспечения на каждой виртуальной машине:

```
ubuntu@ip-172-31-67-190:~$ docker info
   Profile: default
 Kernel Version: 4.4.0-1105-aws
 Operating System: Ubuntu 16.04.6 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 3.858GiB
 Name: ip-172-31-67-190
```

Рисунок 11 – Выполнение команды docker info на Manager (фрагмент)

```
ubuntu@ip-172-31-64-148:~$ docker info
   Profile: default
 Kernel Version: 4.4.0-1105-aws
 Operating System: Ubuntu 16.04.6 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 3.858GiB
 Name: ip-172-31-64-148
```

Рисунок 12 – Выполнение команды docker info на Worker 1 (фрагмент)

```
ubuntu@ip-172-31-69-190:~$ docker info
   Profile: default
 Kernel Version: 4.4.0-1105-aws
 Operating System: Ubuntu 16.04.6 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 3.858GiB
 Name: ip-172-31-69-190
```

Рисунок 13 – Выполнение команды docker info на Worker 2 (фрагмент)

Как мы можем видеть из рисунков 11–13, программное обеспечение Docker успешно установлено на каждой виртуальной машине.

3.3 Запуск кластера Docker Swarm

Для того, что создать кластер, необходимо выполнить команду docker swarm init --advertice-addr [advertise-ip]:2377, где advertise-ip – IP-адрес виртуальной машины, на которой предполагается запуск управляющей ноды [15].

```
ubuntu@ip-172-31-67-190:~$ docker swarm init --advertise-addr 172.31.67.190
Swarm initialized: current node (1axxwd7ck94i60dq8411t0vma) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-34oppkufdai92hzcmcbpjfz75n71nqwhngkmxvrlckx2k17dtu-07npi2zct83i19smhqffzum63 172.31.67.190:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Рисунок 14 – Запуск кластера Docker Swarm

Управляющая нода запущена, кластер готов к добавлению рабочих нод.

Для того, чтобы добавить рабочие ноды в кластер, выполняем команду из вывода результата команды листинга N: docker swarm join –token [token] [manager ip]:[manager port].

```
ubuntu@ip-172-31-64-148:~$ docker swarm join --token SWMTKN-1-34oppkufdai92hzcmcbpjfz75n71nqwhngkmxvrlckx2k17dtu-07npi2zct83i19smhqffzum63 172.31.67.190:2377
This node joined a swarm as a worker.
```

Рисунок 15 – Добавление рабочей ноды 1

```
ubuntu@ip-172-31-69-190:~$ docker swarm join --token SWMTKN-1-34oppkufdai92hzcmcbpjfz75n71nqwhngkmxvrlckx2k17dtu-07npi2zct83i19smhqffzum63 172.31.67.190:2377
This node joined a swarm as a worker.
```

Рисунок 16 – Добавление рабочей ноды 2

Проверим ноды кластера командой: docker node ls.

```
ubuntu@ip-172-31-67-190:~$ docker node ls
ID                            HOSTNAME            STATUS  AVAILABILITY MANAGER STATUS   ENGINE VERSION
pcu632mf2mqwg1rvw2s432ib3     ip-172-31-64-148    Ready   Active                        19.03.9
1axxwd7ck94i60dq8411t0vma *   ip-172-31-67-190    Ready   Active       Leader           19.03.9
zobub3a0zs7i8ygim1oyon1c5     ip-172-31-69-190    Ready   Active                        19.03.9
```

Рисунок 17 – Выполнение команды docker node ls

Из рисунков 14–17 видно, что кластер создан.

Как мы можем видеть на рисунке 17, нода ip-172-31-67-190 является управляющей, а ноды ip-172-31-64-148 и ip-172-31-69-190 являются рабочими.

В конечном счете, мы можем сделать вывод о том, что кластер запущен и готов к проведению исследования.

3.4 Реализация слушателя

Для организации обмена сообщениями используем сетевую утилиту операционной системы Linux netcat.

Создадим слушателя, который будет принимать сообщения и фиксировать время прихода сообщения записью в файл result.txt. Код слушателя продемонстрирован на рисунке 18.

```
#!/bin/bash
ifconfig
while true; do ((nc -l 4789 >> result.txt | exit) && printf "$(date +%s%N | cut
-b1-13) \n" >> result.txt); done
```

Рисунок 18 – Реализация слушателя (server.sh)

3.5 Реализация клиента

С помощью Dockerfile [14] создается контейнер-клиент (client:v2).

Контейнер-клиент выполняет 2 функции:

- устанавливает соединение со слушателем;
- посылает ему сообщение «start».

Код контейнера-клиента (client:v2) продемонстрирован на рисунке 19.

```
FROM ubuntu
RUN apt update && apt-get install -y netcat
CMD ((echo "start") | (nc 172.31.67.190 4789 | exit)) && while true; do echo
"msg"; sleep 1; done;
```

Рисунок 19 – Реализация контейнера клиента (client:v2)

С помощью Dockerfile создается контейнер-клиент (client:v1).

Контейнер-клиент выполняет 2 функции:

- устанавливает соединение с контейнером-слушателем;
- посылает ему сообщение «message».

Код контейнера-клиента (client:v1) продемонстрирован на рисунке 20.

```
FROM ubuntu
RUN apt update && apt-get install -y netcat
CMD ((echo "message") | (nc 172.31.67.190 4789 | exit)) && while true; do echo
"msg"; sleep 1; done;
```

Рисунок 20 – Реализация контейнера клиента (client:v1)

4 ПРОВЕДЕНИЕ ИСПЫТАНИЙ

Для проведения исследований используем утилиту автоматического проведения испытаний с ключами:

– A (испытание «Один контейнер»);

– B (испытание «Группа контейнеров»);

– C (испытание «Горизонтальное масштабирование»).

Продемонстрируем процесс развертывания группы контейнеров на диаграмме развертывания (рисунок 21).

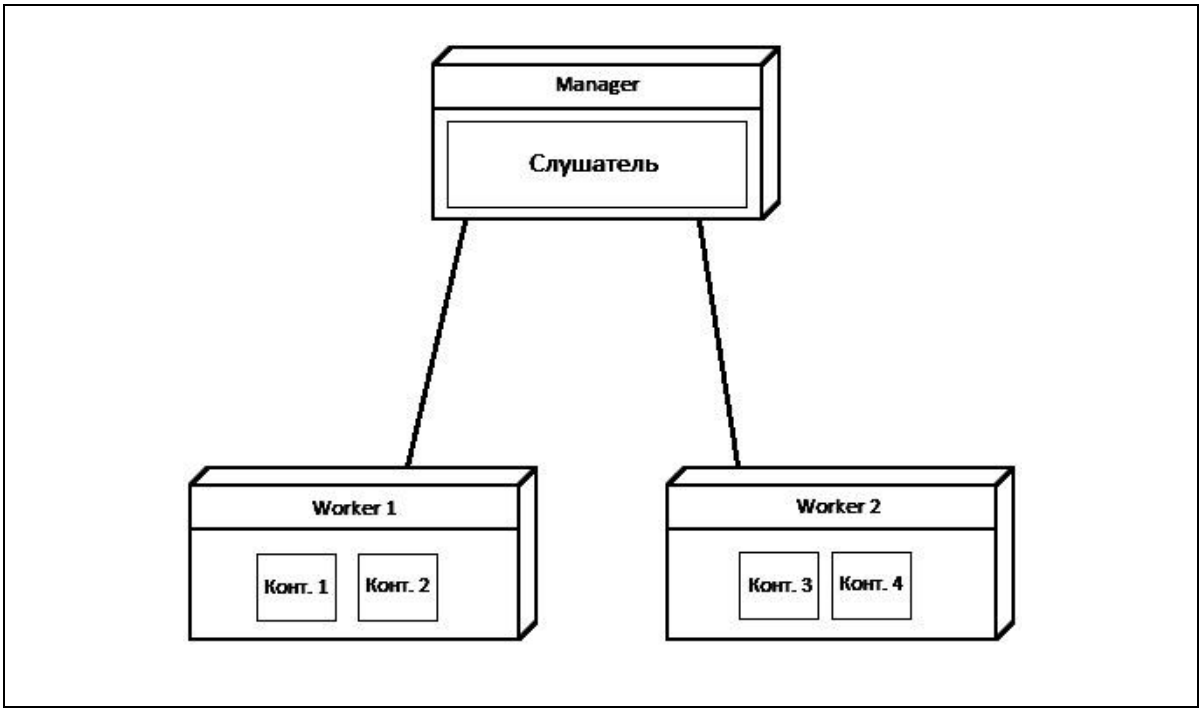

Рисунок 21 – Диаграмма развертывания для испытания "Группа контейнеров"

4.1 Время развертывания одного контейнера в кластере

На рисунках 22–23 можем видеть проведение испытания «Один контейнер» с помощью утилиты автоматического проведения испытаний.

```
ubuntu@ip-172-31-67-190:~$ bash auto-research.sh -A

overall progress: 1 out of 1 tasks
1/1: running
verify: Service converged
```

Рисунок 22 – Развертывание контейнера с помощью утилиты

```
start
1591593712606
message
1591593714458
```

Рисунок 23 – Просмотр данных слушателя

Повторим испытание 10 раз и представим полученные данные в табличном виде (таблица 2).

Таблица 2 – Результаты испытания «Один контейнер»

N	**t_1**	**t_2**	**t_3**	**t_4**	**t_5**	**t_6**	**t_7**	**t_8**	**t_9**	**t_{10}**
1	1852	1260	1866	2480	1072	1773	1130	2496	1857	2216

, где N – количество контейнеров, $t_{1\text{-}10}$ – время выполнения испытания в миллисекундах.

Испытание «Один контейнер» успешно проведено. Результаты получены.

4.2 Время развертывания группы контейнеров в кластере

На рисунках 23–24 можем видеть проведение испытания «Группа контейнеров» с помощью утилиты автоматического проведения испытаний.

```
ubuntu@ip-172-31-67-190:~$ bash auto-research.sh -B 4

overall progress: 4 out of 4 tasks
1/4: running
2/4: running
3/4: running
4/4: running
verify: Service converged
```

Рисунок 24 – Развертывание группы контейнеров с помощью утилиты

```
start
1591594448401
message
1591594449507
message
1591594449833
message
1591594449923
message
1591594450322
```

Рисунок 25 – Просмотр данных слушателя

Повторим каждое испытание 10 раз и представим полученные данные в табличном виде (таблица 3).

Таблица 3 – Результаты испытания «Группа контейнеров»

N	**t_1**	**t_2**	**t_3**	**t_4**	**t_5**	**t_6**	**t_7**	**t_8**	**t_9**	**t_{10}**
2	1814	1793	1797	2384	2009	1800	1761	1139	1904	2168
4	1921	2386	2069	1914	1829	2875	1785	1822	2350	1736
8	2624	3385	2585	2531	3317	2575	2226	2960	2771	2538
16	4805	3728	5300	4217	3719	3906	4251	3973	4219	4793
32	6026	6288	6429	6191	6511	6813	6938	5894	6103	7178
48	8442	9132	7911	8437	8628	8071	7822	8365	7681	8411
64	9693	9903	10774	9766	10535	9943	10811	9836	9678	10234
128	18759	19690	19318	19266	19306	19416	18939	18859	19400	19056
150	23761	23284	23616	22577	23173	23256	23902	22913	23553	23640

, где N – количество контейнеров, $t_{1\text{-}10}$ – время выполнения испытания в миллисекундах.

На рисунке 26 можем видеть, что при увеличении количества развертываемых контейнеров до 180, управляющая нода не справляется с данной задачей из-за нагрузки на ядра процессора, которая достигает 100%, что приводит к принудительному завершению команды развертывания.

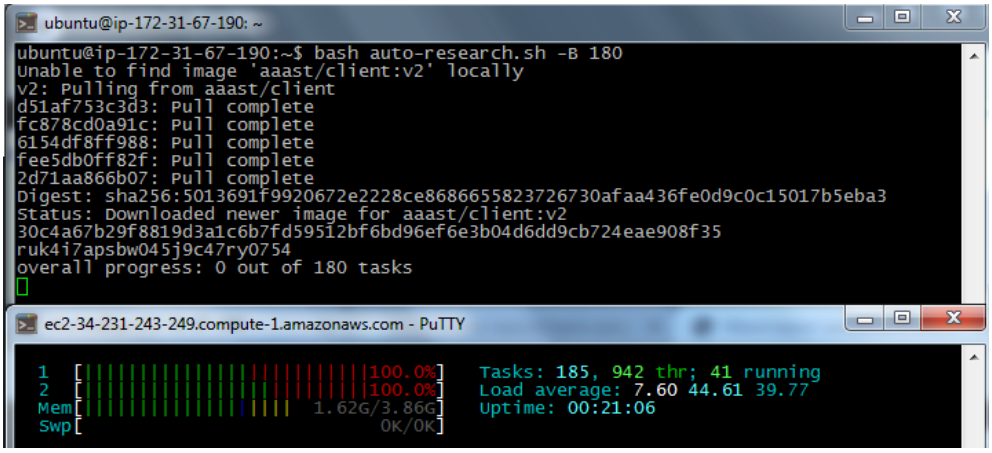

Рисунок – 26. Развертывание 180 контейнеров.

4.3 Время горизонтального масштабирования существующих контейнеров

На рисунках 27–28 можем видеть проведение испытания «Горизонтальное масштабирование» с помощью утилиты автоматического проведения испытаний.

```
ubuntu@ip-172-31-67-190:~$ bash auto-research.sh -C 1 16
kv5smel0swnfqp26wzub2dke0
overall progress: 1 out of 1 tasks
1/1: running
verify: Service converged

my-client scaled to 16
overall progress: 16 out of 16 tasks
1/16: running
2/16: running
3/16: running
4/16: running
5/16: running
6/16: running
7/16: running
8/16: running
9/16: running
10/16: running
11/16: running
12/16: running
13/16: running
14/16: running
15/16: running
16/16: running
verify: Service converged
```

Рисунок 27 – Горизонтальное масштабирование с помощью утилиты

```
message
1591598241328

start
1591598247980
message
1591598249379
message
1591598249475
message
1591598249707
message
1591598249882
message
1591598249907
message
1591598249982
message
1591598250150
message
1591598250211
message
1591598250475
message
1591598250570
message
1591598250719
message
1591598250803
message
1591598251066
message
1591598251086
message
1591598251570
message
1591598251978
```

Рисунок 28 – Просмотр данных слушателя

Повторим каждое испытание 10 раз и представим полученные данные в табличном виде (таблица 4).

Таблица 4 – Результаты испытания «Горизонтальное масштабирование»

N-M	$\mathbf{t_1}$	$\mathbf{t_2}$	$\mathbf{t_3}$	$\mathbf{t_4}$	$\mathbf{t_5}$	$\mathbf{t_6}$	$\mathbf{t_7}$	$\mathbf{t_8}$	$\mathbf{t_9}$	$\mathbf{t_{10}}$
1-16	4043	4028	3073	3922	3914	3693	4388	3089	3338	3590
16-64	7525	6972	7770	7636	7837	7079	8734	6912	6772	6279

, где N – исходное количество контейнеров, M – конечное количество контейнеров, $t_{1\text{-}10}$ – время выполнения испытания в миллисекундах.

Испытание «Горизонтальное масштабирование» успешно проведено. Результаты получены.

4.4 Временные задержки на передачу сообщений между сервисами в кластере

На рисунках 29–37 можем видеть проведение испытания «Latency».

```
ubuntu@ip-172-31-64-148:~$ docker network create \
>   --driver overlay \
>   --subnet 10.0.9.0/24 \
>   --opt encrypted \
>   my-network
toq9fcmcbwwyq2ol1stjv9t47
```

Рисунок 29 – Создание сети

```
ubuntu@ip-172-31-64-148:~$ docker network ls
NETWORK ID   NAME            DRIVER   SCOPE
0eba6ee9f6da bridge          bridge   local
cb519c9b6ceb docker_gwbridge bridge   local
8abea94289d5 host            host     local
c0bi3i7g366b ingress         overlay  swarm
toq9fcmcbwwy my-network      overlay  swarm
e509e9407e6f none            null     local
```

Рисунок 30 – Проверка сети

```
ubuntu@ip-172-31-64-148:~$ docker service create \
>   --replicas 3 \
>   --name my-web \
>   --network my-network \
>   nginx
vkl22s0fq8gk8u2htkje8632s
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service converged
```

Рисунок 31 – Создание и подключение первого сервиса к сети

```
ubuntu@ip-172-31-64-148:~$ docker service inspect \
>   --format='{{json .Endpoint.VirtualIPs}}' \
>   my-web
[{"NetworkID":"toq9fcmcbwwyq2ol1stjv9t47","Addr":"10.0.9.2/24"}]
```

Рисунок 32 – Нахождение IP-адреса первого сервиса

```
ubuntu@ip-172-31-64-148:~$ docker service create --replicas 3 --name my-web2 --network my-network nginx
adtfg4qvpt0r2c20yasc5ac0u
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
```

Рисунок 33 – Создание и подключение второго сервиса к сети

```
ubuntu@ip-172-31-64-148:~$ docker service inspect --format='{{json .Endpoint.VirtualIPs}}' my-web2
[{"NetworkID":"toq9fcmcbwwyq2ol1stjv9t47","Addr":"10.0.9.8/24"}]
```

Рисунок 34 – Нахождение IP-адреса второго сервиса

```
ubuntu@ip-172-31-64-148:~$ docker exec -it 60e8d8b04bff bash
```

Рисунок 35 – Подключение к сервису

```
root@60e8d8b04bff:/# apt-get install iputils-ping
```

Рисунок 36 – Установка iputilits-ping

```
root@60e8d8b04bff:/# ping -c 10 -i 0.1 -v 10.0.9.2
PING 10.0.9.2 (10.0.9.2) 56(84) bytes of data.
64 bytes from 10.0.9.2: icmp_seq=1 ttl=64 time=0.127 ms
64 bytes from 10.0.9.2: icmp_seq=2 ttl=64 time=0.112 ms
64 bytes from 10.0.9.2: icmp_seq=3 ttl=64 time=0.108 ms
64 bytes from 10.0.9.2: icmp_seq=4 ttl=64 time=0.096 ms
64 bytes from 10.0.9.2: icmp_seq=5 ttl=64 time=0.109 ms
64 bytes from 10.0.9.2: icmp_seq=6 ttl=64 time=0.109 ms
64 bytes from 10.0.9.2: icmp_seq=7 ttl=64 time=0.111 ms
64 bytes from 10.0.9.2: icmp_seq=8 ttl=64 time=0.110 ms
64 bytes from 10.0.9.2: icmp_seq=9 ttl=64 time=0.108 ms
64 bytes from 10.0.9.2: icmp_seq=10 ttl=64 time=0.110 ms

--- 10.0.9.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 936ms
rtt min/avg/max/mdev = 0.096/0.110/0.127/0.007 ms
```

Рисунок 37 – Выполнение команды ping –c 10 –i 0.1 –v 10.0.9.2

Представим полученные данные в табличном виде (таблица 5).

Таблица 5 – Результаты испытания «Latency»

N	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
10	0.127	0.112	0.108	0.096	0.109	0.109	0.111	0.110	0.108	0.110

, где N – количество запросов, t_{1-10} – время выполнения запросов в миллисекундах.

Испытание «Latency» успешно проведено. Результаты получены.

4.5 Вывод

Проанализируем полученные данные из ранее проведенных испытаний и представим полученные значения в табличном виде (таблица 6).

Таблица 6 – Итоговые результаты испытаний

Наименование испытаний	**N**	**Средние время, мс**	**Среднее время развертывания одного контейнера, мс**
Время развертывания одного контейнера в кластере	1	1 800,2	1 800,20
Время развертывания группы контейнеров в кластере	2	1 856,9	928,45
	4	2 068,7	517,18
	8	2 751,2	343,90
	16	4 291,1	268,19
	32	6 437,1	201,16
	48	8 290,0	172,71
	64	10 117,3	158,08
	128	19 255,6	150,43
	150	23 533,6	157,02
	180	N/A	N/A
Время горизонтального масштабирования существующих контейнеров	1-16	3 707,8	247,19
	16-64	7 351,6	153,16
Временные задержки на передачу сообщений между хостами в кластере	10	0,11	N/A

Построим графики полученных результатов.

Рисунок 38 – Среднее время развертывания

Рисунок 39 – Среднее время развертывания одного контейнера

Из рисунка 38 можем видеть, что с увеличением количества развертываемых контейнеров время развертывания увеличивается.

Так, допустим, при увеличении количества контейнеров с 1 до 16, время развертывания увеличивается в 2,4 раза, при увеличении количества контейнеров с 16 до 64, время развертывания также увеличивается в 2,4 раза, при увеличении количества с 64 до 150, время развертывания увеличивается в 2,3 раза.

На рисунке 39 наблюдаем, что при увеличении числа развертываемых контейнеров среднее время развертывания одного контейнера уменьшается. Так, для развертывания одного контейнера требуется 1800,2 мс, когда для группы из 64 контейнеров, среднее время развертывания одного контейнера занимает всего 158,08 мс. Снижение времени развертывания одного контейнера при увеличении количества контейнеров связано с распараллеливанием выполнения задач развертывания в кластере.

Также стоит обратить внимание на то, что при развертывании группы из 128 контейнеров время развертывания одного контейнера занимает 150,01 мс, когда для группы из 150 контейнеров время развертывания занимает 155,78 мс, что говорит нам о том, найдена точка насыщения среднего времени развертывания одного контейнера при выполнении задачи развертывания группы контейнеров.

Отдельно стоит отметить, что при увеличении количества развертываемых контейнеров до 180, управляющая нода не справляется с данной задачей из-за нагрузки на ядра процессора, которая достигает 100%, что приводит к принудительному завершению команды развертывания.

Из таблицы 6 можем заметить, что масштабировать контейнеризированные приложения выгоднее, чем развертывать необходимое количество заново, так как при эквивалентных задачах (развертывание 16

контейнеров и горизонтальное масштабирование с 1 до 16 контейнеров, а также развертывание 48 контейнеров и горизонтальное масштабирование с 16 до 64 контейнеров) видим, что масштабирование выполняется быстрее примерно на 12-14% за счет того, что масштабируемое контейнеризованное приложение не нуждается в дополнительном скачивании образов.

Среднее время задержки при передаче запросов между кластеризованными узлами составило 0.11 мс.

Так как лидером в программном обеспечении для автоматизации развертывания и управления приложениями в средах с поддрежкой контейнеризации является Docker, а Docker Swarm является родной системой кластеризации, которая превращает набор Docker-хостов в один последовательный кластер, можем отметить беспрепятственное взаимодействие кластера Docker Swarm с Docker-контейнерами.

Также стоит отметить, что платформа контейнерной оркестрации Docker Swarm отличается простотой освоения и развертывания кластера.

5. ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была разработана утилита автоматического проведения испытаний. Также были проведены испытания «Один контейнер», «Группа контейнеров», «Горизонтальное масштабирование», «Latency».

Для выполнения поставленной цели были решены следующие задачи:

- произведен подбор литературы, научных публикаций и интернет статей, необходимых для проведения исследования;
- выполнен обзор платформ контейнерной оркестрации;
- определены ключевые требования и критерии проведения исследования;
- спроектирована и реализована утилита автоматического проведения испытаний;
- выполнено исследование эффективности платформ контейнерной оркестрации при организации туманных вычислений;
- проанализированы полученные результаты и сделаны сопутствующие выводы.

Результаты работы были доложены на 73 студенческой научной конференции ЮУрГУ.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. IOT, туман и облака: поговорим про технологии. [Электронный ресурс] URL: https://habr.com/ru/company/cloud4y/blog/467711/ (дата обращения: 18.05.2020)
2. Зачем облака спускаются на землю. [Электронный ресурс] URL: https://mcs.mail.ru/blog/why-clouds-get-closer-to-the-ground (дата обращения: 18.05.2020)
3. Довгаль, В.А. Роль туманных вычислений в Интернете Вещей / Ежеквартальный рецензируемый, реферируемый научный журнал «Вестник АГУ». Выпуск 4(231), 2018. – С. 205–209.
4. Hoque, S. Towards Container Orchestration in Fog Computing Infrastructures / S. Hoque, M.S. de Brito, T. Magedanz, A. Willner, O. Keil // IEEE 41st Annual Computer Software and Applications Conference. – 2017. – P. 294–299.
5. Virtualization Technology & Virtual Machine Software: What is Virtualization? [Электронный ресурс] URL: https://www.vmware.com/ru/solutions/virtualization.html (дата обращения: 18.05.2020)
6. Ажиотаж вокруг контейнеров. [Электронный ресурс] URL: https://www.osp.ru/lan/2014/10/13043208/ (дата обращения: 18.05.2020)
7. Гордеев, А.В. Сравнительное тестирование контейнерной и гипервизорной виртуализации / А.В. Гордеев, Д.В. Горелик, СПбГУАП, Информационно-управляющие системы №2, 2018. – С. 60–66.
8. Упакуем все – зачем нужны контейнеры и как с ними работать в Big Data. [Электронный ресурс] URL: https://medium.com/@bigdataschool/упакуем-все-зачем-нужны-контейнеры-и-как-с-ними-работать-в-big-data-256acd15fd8f

9. Зачем и как использовать контейнеры – разбираемся с Docker, Kubernetes и другими инструментами. [Электронный ресурс] URL: https://tproger.ru/articles/containers-explained/amp/ (дата обращения: 18.05.2020)
10. Моуэт, Э. Использование Docker / Э. Моуэт; пер. с англ. А.В. Снастина; науч. ред. А. А. Маркелов. – М.: ДМК Пресс, 2017. – 354 с.
11. de Brito, M. S. A Service Orchestration Architecture for Fog-enabled Infrastructures / M. S. de Brito, S. Hoque, T. Magedanz R. Steinke, A. Willner, D. Nehls, O. Keils, F. Schreiner // 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC). – 2017. – P. 127–132.
12. Khan A. Key Characteristics of a Container Orchestration Platform to Enable a Modern Application / IEEE Cloud computing. – 2017. – P. 42–48.
13. Evaluating Container Platforms at Scale. [Электронный ресурс] URL: https://medium.com/on-docker/evaluating-container-platforms-at-scale-5e7b44d93f2c (дата обращения: 29.05.2020)
14. Электронная документация Docker. [Электронный ресурс] URL: https://docs.docker.com (дата обращения: 19.03.2020)
15. Электронная документация Docker Swarm. [Электронный ресурс] URL: https://docs.docker.com/engine/swarm/ (дата обращения: 19.03.2020)