Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО»

(Университет ИТМО)

Факультет **БИТ**

Образовательная программа Информационная безопасность / Information Security 2022

Направление подготовки (специальность) 10.04.01 Информационная безопасность

О Т Ч Е Т

о научно-исследовательской работе

Наименование темы: **Обеспечение доступности высоконагруженных приложений с помощью Kubernetes**

Обучающийся: **Громов Артем Андреевич, магистрант группы № N4140c**

Согласовано:
Научный руководитель: **Югансон Андрей Николаевич, к.т.н., доцент ФБИТ, Университет ИТМО**
Ответственный за научно-исследовательскую работу: **Заколдаев Данил Анатольевич, к.т.н., декан ФБИТ, Университет ИТМО**

Научно-исследовательская работа выполнена с оценкой____

оп. А.М. Югансон

Дата 20.01.23

Санкт-Петербург
2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Основной целью данной научно-исследовательской работы является формирование и определение основного направления магистерского диссертационного исследования.

Для достижения поставленной цели в рамках данного исследования предполагается выполнение следующих задач:

1) Изучить общепринятые стандарты обеспечения отказоустойчивости в различных информационных системах.
2) Изучить текущие практики обеспечения высокой доступности с помощью Kubernetes.
3) Провести обзор предметной области исследований.
4) Провести аналитический обзор существующих научных работ в данной сфере.

В рамках реализуемой научно-исследовательской работы рассматриваются основные способы обеспечения отказоустойчивости приложений. Приложения могут быть запущены, как с помощью средств полной виртуализации, так и с помощью частичной виртуализации (ОС-виртуализации, контейнеризации). В частности, рассмотрены средства высокой доступности (HA) от таких вендоров, как VMware и Citrix XenServer [1]. А также оценены актуальные возможности обеспечения отказоустойчивости с помощью Docker и Kubernetes[2, 3, 4]. Необходимо отметить, что не существует системы способной работать без остановки 100% времени в год. Однако существующют решения, обеспечивающие до 99,9999% времени работы.

ПРОБЛЕМА АКТУАЛЬНОСТИ ОБЕСПЕЧЕНИЯ ДОСТУПНОСТИ ВЫСОКОНАГРУЖЕННЫХ ПРИЛОЖЕНИЙ С ПОМОЩЬЮ KUBERNETES

В настоящее время большая часть жизни современного человека проходит в интернете: работа, обучение (высшие учебные заведения, курсы повышения квалификации), онлайн торговля (одежда, электроника, стройматериалы и тд), взаимодействие с государственными службами (налоговая, госуслуги). В связи с этим появляется проблема - отсутствие доступности этих приложений и сервисов может сильно повлиять на ритм жизни человека. А с переносом критически важных сервисов (онлайн-банкинг, госуслуги) в онлайн, а также с постоянным увеличением количества пользователей этих сервисов, остро встает вопрос о повышении качества обеспечения отказоустойчивости.

Отсутствие доступности критически важных сервисов более 14 минут в день (соответствует доступности 99%) может привести к большим отрицательным последствиям. На мой взгляд, такие сервисы должны стремить обеспечивать доступность на уровне 99,99% (соответствует отказу работы сервиса в течение 8.64 секунд в день). В тоже время для всех остальных сервисов отказоустойчивость, соответствующая 99,5% (7 минут 12 секунд в день) будет вполне достаточно.

Для достижения отказоустойчивости такого уровня используются различные механизмы: высокая доступность (High Availability), отказоустойчивость (Fault Tolerance), непрерывная доступность (Continuous availability), восстановление после катастроф (Disaster recovery). В данной научно-исследовательской работе будет подробно рассмотрен механизм позволяющий обеспечивать высокую доступность (HA).

ВЫВОДЫ ПО ПЕРВОЙ ГЛАВЕ

Повышение доступности онлайн сервисов и приложений крайне важная задача, как с точки зрения пользовательского опыта, так и с точки

зрения информационной безопасности. Вне зависимости от критичности сервиса, его отказоустойчивость является одним из главных свойств.

В первой главе были перечислено множество методов повышения отказоустойчивости, однако основной упор в данной работе будет сделан на внедрение механизма высокой доступности (HA). Данный выбор связан с тем, что на момент написания этой научно-исследовательской работы существует множество инструментов, позволяющих достаточно быстро и без серьезных временные и финансовых вложений повысить уровень предоставляемой отказоустойчивости именно с помощью механизма высокой доступности (HA).

СРАВНЕНИЕ ОБЕСПЕЧЕНИЯ ВЫСОКОЙ ДОСТУПНОСТИ С ПОМОЩЬЮ ВИРТУАЛЬНЫХ МАШИН И КОНТЕЙНРОВ

Обеспечение высокой доступности с помощь виртуальных машин

Недавнее развитие и оптимизация технологий виртуализации в последние десятилетия привели к их широкому внедрению, и растущей тенденции к размещению рабочей нагрузки на виртуализированных платформах. Хотя технологии виртуализации помогают в снижение стоимости и сложности за счет выхода на новый уровень абстракции разработчикам нет необходимости заботиться о физических ресурсах, они также вызывают опасения по поводу доступности приложений, размещенных на виртуализированных платформах. В частности, большое количество виртуальных машин на одном хосте может негативно сказаться на доступности приложений, поскольку все виртуальные машины, а также услуги, предоставляемые этими машинами, выйдут из строя в случае отказа хоста.

Чтобы гарантировать высокую доступность приложений (HA), несмотря на катастрофические и разрушительные события, в технологии виртуализации интегрируются такие функции HA, как мониторинг виртуальных машин, живая миграция (live migration) и контрольная точка/восстановление (checkpoint/restore). В этой научно-исследовательской работе будет проведено сравнение технологий виртуализации с точки зрения HA. Современные решения HA в двух основных типах виртуализированных платформ: платформа на базе гипервизора и платформа на базе контейнеров. Соответственно исследуются с точки зрения ограничений и возможностей HA.

Высокая доступность (HA) относится к измерению способности системы (или приложения, компонента) оставаться постоянно функциональной 99,999% времени (также известной как пять девяток) [12]. Некоторые системы, такие как система управления воздушным движением и система экстренного реагирования (9-1-1), предъявляют еще более жесткие

требования к доступности, а именно отказоустойчивости (FT), или непрерывности предоставления услуг.

Приложения без отслеживания состояния, способные хранить свое состояние в реплицированном распределенном хранилище, могут быть развернуты за балансировщиками нагрузки, которые в случае потери доступа перенаправят трафик на другие работоспособные дубликаты. Тем не менее, для приложений с отслеживанием состояния, где необходимо поддерживать и состояние приложения и сетевого стека, требуется другой подход.

Для достижения HA в виртуализированной среде платформам виртуализации необходимы следующие функции:

1) возможность получения и сохранения состояния ВМ;
2) возможность осуществления живой миграции ВМ.

В дополнение к ним, полное решение HA должно включать обязательные функции:

1) непрерывное обнаружение сбоев;
2) автоматическую синхронизацию состояния между активным и резервным сервером;
3) управление восстановлением путем динамического перехода на резервный сервер при обнаружении сбоя и прекращении работы активного.

Поставщики средств виртуализации предлагают своим клиентам различные решения HA/FT. Обычно среди этих решений HA реализуется путем встраивания в систему нескольких уровней возможностей сопротивления отказу. Типичное решение HA состоит из набора слабосвязанных серверов, которые являются автономными и постоянно контролируют работоспособность (например, через heartbeat) друг друга. В случае отказа хоста виртуальные машины могут переходить с одного сервера на другой. Для целей FT, однако, недостаточно только переключения рабочей нагрузки. Чтобы гарантировать непрерывность обслуживания, требуется наличие дубликата (горячего резерва), который должен быть тесно связан и согласован с основным приложением, чтобы в случае отказа, дубликат всегда

был готов взять на себя управление без прерывания обслуживания и потери данных.

Хотя решения HA/FT могут значительно отличаться по реализации, они имеют один и тот же принцип - дублирование критически важных компонентов с помощью избыточности в попытке устранить единые точки отказа.

В таблице 1 представлен краткий обзор возможностей HA двух основных поставщиков, т.е. VMware и Citrix XenServer. В целом, HA поддерживается всеми из них.

Как показано в таблице 1, живая миграция виртуальных машин поддерживается как VMware, так и XenServer с помощью vMotion и XenMotion соответственно. Однако для того, чтобы ВМ могла нормально работать на резервном хосте после миграции, требуется совместимость процессоров. Ожидается, что процессоры на основном и резервном хостах будут обеспечивать одинаковый набор функций для виртуальных машин, чтобы основная виртуальная машина и сопутствующие приложения не выходили из строя. Между тем, контрольная точка/восстановление (Checkpoint/Restore) также поддерживается всеми из них, обеспечивая возможность создания снимков виртуальных машин.

Отказоустойчивость (FT) может использовать несколько механизмов для обеспечения доступности: контрольные точки (checkpointing) и запись и воспроизведение (Record-and-Replay). В отличие от высокой доступности (HA), которая обычно достигается с помощью отказоустойчивой кластеризации, отказоустойчивость виртуальных машин является более сложной задачей в виртуализированных платформах, поскольку эффективная синхронизация резервной ВМ с основной является сложной задачей. В настоящее время существует две основные стратегии решения этой проблемы.

Первая - запись и воспроизведение (Record-and-Replay), которая записывает все входные данные на основной ВМ, отправляет их по

выделенному каналу на резервную копию и затем воспроизводит их на ней. Реализация этой стратегии для однопроцессорной ВМ сравнительно проста, поскольку все инструкции, выполняемые vCPU в основной ВМ, можно с точностью воспроизвести на vCPU в резервной ВМ. Однако, с точки зрения производительности, в современных архитектурах центральные процессоры обычно содержат несколько ядер и используют такие методы, как предсказание ветвлений, спекуляции и внеочередное выполнение, что вносит неопределенность в выполнение программ. Эта неопределенность увеличивает сложность синхронизации резервной копии с основной. Эту проблему часто называют отказоустойчивостью симметричных многопроцессорных систем (SMP FT). Сотрудники компании Intel утверждают, что эффективная система записи и воспроизведения, нацеленная на решение этой задачи, должна быть обеспечена аппаратной поддержкой. Однако, VMware внедрили поддержку SMP FT в шестой версии. По этой причине VMware, может обеспечить FT как для однопроцессорных ВМ, так и для многопроцессорных. Также Marathon everRun MX имеет решение этой проблемы.

Вместо записи входов в основную ВМ, альтернативная стратегия (т.е. контрольная точка (checkpointing). Данный механизм проверяет состояние основной ВМ после того, как произошли какие-либо изменения, после чего, отправляет все изменения на резервную машину, и поддерживает постоянную синхронизацию резервной ВМ с основной. В отличие от записи и воспроизведения (Record-and-Replay), контрольные точки имеют свои преимущества в простоте и поддержке SMP. Однако производительность данного решения сильно зависит от частоты создания контрольных точек и объема данных, которые необходимо проверить и передать на резервную машину. Remus и Kemari являются реализациями такого подхода.

Таблица 1 - Сравнение технологий виртуализации с точки зрения HA

Функции	VMware	Citrix XenServer	Docker + Kubernetes
Отказоустойчивость	Полная с поддрежкой многопроцессорности	Полная с поддрежкой многопроцессорности	Да
Живая миграция	Да, vMotion, необходима совместимость процессоров	Да, XenMotion, необходима совместимость процессоров	Да, с помощью механизмов Docker
Контрольная точка/восстановление	Да	Да	Да, с помощью механизмов Docker
Обнаружение сбоев	Автоматическое	Автоматическое	Да
Управление отказоустойчивостью	да	да	Да
Гостевая ОС	Любая	Любая	Linux

Обеспечение высокой доступности с помощь контейнеров

В отличие от виртуализации на базе гипервизоров, виртуализация на базе контейнеров (она же виртуализация на уровне операционной системы) не направлена на эмуляцию всей аппаратной среды, а позволяет современному ядру Linux управлять изоляцией между приложениями. При виртуализации на уровне ОС несколько изолированных систем Linux (контейнеров) могут работать на управляющем хосте, разделяя один экземпляр ядра. Каждый контейнер имеет свой собственный процесс и сетевое пространство. Такие системы, как LXC (LinuX Containers) и Docker, являются примерами реализации этого типа.

По сути, контейнер — это набор процессов (как правило, связанных с хранением данных), полностью изолированных от других контейнеров. Для обеспечения высокой доступности контейнера необходимо наличие возможности контрольной точки/восстановления процесса. На решение этой задачи было направлено большое количество усилий, включая BLCR [9], DMTCP [1] и ZAP [23]. Однако эти системы не были специально разработаны для контейнеров. В них либо отсутствует та или иная функция,

либо они обычно поддерживают только ограниченный набор приложений. Более того, ни одна из них не стала частью основного ядра Linux из-за сложности реализации.

Благодаря добавлению высокоуровневых функций, таких как версионирование и совместное использование, поверх LXC+ Docker становится привлекательной платформой для размещения контейнеров. А с появлением Kubernetes и CRIU виртуализация на уровне операционной системы обрела все механизмы для обеспечения непрерывной работы: высокая доступность(High Availability), отказоустойчивость (Fault Tolerance), живая миграция (Live migration)[5], контрольная точка/восстановление (Checkpoint/Restore)[2], обнаружение сбоев, управление отказоустойчивостью[5, 6].

Рассмотрим подробно один из механизмов для обеспечения отказоустойчивости с помощью контейнеризации: контрольная точка/восстановление (Checkpoint/Restore).

Подход к использованию механизма контрольная точка/восстановление для Docker с CRIU можно разделить на две части: контрольную точку и восстановление.

Процедура создания контрольной точки в значительной степени полагается на директорию в файловой системе: /proc (это общее место, где criu берет всю необходимую информацию). Сюда входит информация о дескрипторах файлов, параметрах каналов и картах распределения памяти. Для выполнения процедуры создания контрольной точки выполняются следующие шаги:

1) Собирается дерево процессов после чего замораживается. Используя команду $pid, CRIU проходит через каталог /proc/$pid/task/, собирая потоки, и через /proc/$pid/task/$tid/children, чтобы рекурсивно собрать дочерние процессы. Во время прохождения, задачи останавливаются с помощью команды ptrace: PTRACE_SEIZE.

2) Сбор ресурсов задач. На этом этапе CRIU считывает всю известную ему информацию о собранных задачах и записывает ее. Затем CRIU внедряет код паразита в задачу через интерфейс ptrace.

3) Очистка. После того, как все сброшено (например, страницы памяти, которые могут быть записаны только изнутри сброшенного адресного пространства), CRIU снова используем средство ptrace и очищает собранную информацию, удаляя весь лишний код-паразит и восстанавливая оригинальный код. Затем CRIU отсоединяется от задач, и они продолжают работать.

Процедура восстановления выполняется путем превращения CRIU в задачи, которые этот механизм восстанавливает. На верхнем уровне эта процедура состоит из 4 шагов:

1) Определить общие ресурсы. На этом этапе CRIU считывает файлы образов и выясняет, какие процессы совместно используют какие ресурсы. В дальнейшем общие ресурсы восстанавливаются одним процессом, а все остальные либо наследуют один на 2 этапе (как сессия), либо получают другим способом.

2) Отделение дерева процессов. На этом этапе CRIU вызывает функцию fork() много раз для повторного создания процессов, которые необходимо восстановить.

3) Восстановить ресурсы основных задач. Здесь CRIU восстанавливает все ресурсы, кроме точного отображения памяти, таймеров, мандатов, потоков. На этом этапе CRIU открывает файлы, подготавливает пространства имен, сопоставляет (и заполняет данными) частные области памяти, создает сокеты, вызывает функции chdir() и chroot() и делает еще несколько операций.

4) Переключение на контекст восстановления, восстановление оставшихся элементов и продолжение функционирования. Поскольку механизм CRIU превращается в основной процесс, ему придется расформировать всю свою память и вернуть память скопированного

процесса. В процессе выполнения этого действия некоторый код должен существовать в памяти (код, выполняющий munmap и mmap). Там же, где мы восстанавливаем таймеры, чтобы они не срабатывали слишком рано, полномочия, чтобы позволить CRIU выполнять привилегированные операции (например, fork-with-pid), и потоки, чтобы они не подверглись внезапному изменению структуры памяти.

ВЫВОДЫ ПО ВТОРОЙ ГЛАВЕ

Проанализировав научные статьи из предметной области, можно сделать вывод, что технологии высокой доступности (HA) применяются долгое время в полной виртуализации. Однако, виртуализация на основе операционной системы все еще требует улучшений для достижения большей стабильности.

ЗАКЛЮЧЕНИЕ

В ходе выполнения исследования в рамках научно-исследовательской работы, были выполнены в полном объёме цель и поставленные задачи. А также, выполнено в полном объёме индивидуальное задание в соответствии с графиком выполнения работ.

Планируемые результаты научно-исследовательской работы получены полностью и собраны в качестве необходимого материала для реализации дальнейшего исследования в рамках магистерской выпускной квалификационной работы.

На данный момент, в связи с повышенным вниманием к российским онлайн сервисам и приложениям, повышение их безопасности является приоритетной задачей для любой организации, вне зависимости от того: государственная она или частная. В данной работе, а также в магистерском диссертационном исследовании, фокус будет направлен на один из аспектов информационной безопасности - обеспечение доступности онлайн сервисов и приложений.

Проанализировав предметную область предполагаемого исследования, а также изучив некоторое количество научных работ в данной сфере (проанализировано и подобрано 6 научных работ), можно подтвердить, что обеспечение отказоустойчивости онлайн сервисов и приложений является актуальной проблемой.

Так как многие новые, современные приложения используют микросервисную архитектуру, использование контейнеров – наиболее эффективный способ разработки, тестирования и доставки продукта конечному пользователю. По этой причине использование Kubernetes необходимо для обеспечения высокой доступности данных приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Wubin Li, Ali Kanso Comparing Containers versus Virtual Machines for Achieving High Availability // IEEE International Conference on Cloud Engineering. - 2015. - С. 353-358.

2 Chen Yang Checkpoint and Restoration of Micro-service in Docker Containers // Proceedings of the 3rd International Conference on Mechatronics and Industrial Informatics. - 2015. - С. 915-918.

3 Yikang Liu High Availability of Network Service on Docker Container // Proceedings of the 2016 5th International Conference on Measurement, Instrumentation and Automation (ICMIA 2016). – 2016

4 A. A. Khatami, Y. Purwanto and M. F. Ruriawan, High Availability Storage Server with Kubernetes // 2020 International Conference on Information Technology Systems and Innovation (ICITSI). - 2020. - С. 74-78.

5 Schrettenbrunner, Jakob Migrating Pods in Kubernetes. // 2020

6 Vayghan, Leila & Saied, Mohamed & Toeroe, Maria & Khendek, Ferhat. Kubernetes as an Availability Manager for Microservice Applications. // 2019