МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра дискретной математики и информационных технологий

РАЗРАБОТКА СИСТЕМЫ НЕПРЕРЫВНОЙ ПОСТАВКИ ОБНОВЛЕНИЙ ДЛЯ МИКРОСЕРВИСОВ С ВЫДЕЛЕННОЙ БАЗОЙ ДАННЫХ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 421 группы
направления 09.03.01 — Информатика и вычислительная техника
факультета КНиИТ
Хабибуллина Ильи Руслановича

Научный руководитель
доцент, к. т. н. _____ А. О. Соколов

Заведующий кафедрой
доцент, к. ф.-м. н. _____ Л. Б. Тяпаев

Саратов 2019

ВВЕДЕНИЕ

В разработке WEB-приложений, часто используется архитектура построения приложений MVC – Model View Controller. При таком подходе к построению приложений внутри одного сервиса находятся бизнес-логика, графический пользовательский интерфейс и доступ к данным, также такие системы называются монолитными. Применяя такой подход, добавляя новые функции и изменяя старые, система начинает усложняться, становится тяжелее добавлять новые функции, тяжелее поддерживать, тестировать и масштабировать. В качестве альтернативы данной архитектуре используется микросервисная архитектура. Микросервисная архитектура – это архитектурный стиль, в котором приложение представлено в виде набора простых в поддержке и тестировании, слабосвязанных, независимо разворачиваемых, организованных вокруг возможностей бизнеса сервисов. Микросервисы также позволяют организации расширять свой технологический стек. Однако при работе с такой архитектурой возникают некоторые проблемы, например, как правильно разбить систему на микросервисы и проблема поставки. В данной работе будет рассмотрена проблема поставки микросервиса с выделенной базой данных.

Целью данной дипломной работы является разработка системы непрерывной поставки обновлений для микросервиса с выделенной базой данных. Для достижения этих целей поставлены следующие задачи:

1. изучить технологии непрерывной интеграции и непрерывного разворачивания программного обеспечения, технологии контейнеризации и оркестровки контейнеров;
2. разработать конвейер непрерывной поставки обновлений для микросервиса с выделенной базой данных.

Структура и объём работы. Бакалаврская работа состоит из введения, 4 разделов, заключения и списка использованных. Общий объем работы – 54 страниц, из них 48 страниц – основное содержание, включая 25 рисунков цифровой носитель в качестве приложения, список использованных источников информации – 20 наименований.

1 КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Непрерывная интеграция и непрерывная поставка микросервисов» посвящен теоретическим основам создания приложений по принципам микросервисной архитектуры и построения необходимого для их правильного функционирования CI/CD.

Непрерывная интеграция (англ. Continuous Integration) – это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем. Процесс непрерывной интеграции принято разделять на стадии, например, на следующие: стадия получения исходного кода, стадия сборки, стадия тестирования. Это имеет смысл, поскольку если код не проходит сборку, то не имеет смысла запускать на нем тесты. Совокупность стадий называется конвейером (Pipeline).

В разработке микросервисов важно, чтобы можно было внести изменение в отдельный микросервис и развернуть его независимо от остальных микросервисов. Есть несколько вариантов как это можно реализовать. В первом варианте весь код находится в одном большом хранилище. Любая проверка этого исходного кода запустит сборку, которая запустит все этапы верификации, связанные с микросервисами, и создаст несколько артефактов, имеющих обратную связь с той же самой сборкой.

В данном разделе приводится сравнение двух подходов к применению процесса непрерывной интеграции и непрерывной поставки микросервисов. В первом, код всех микросервисов хранится в одном хранилище. Любая проверка этого исходного кода запустит сборку, которая запустит все этапы верификации, связанные с микросервисами, и создаст несколько артефактов, имеющих обратную связь с той же самой сборкой. Преимущества такого подхода: весь код находится в одном репозитории, простое создание сборок. Недостатками такого подхода является: плохая масштабируемость и низкая скорость разворачивания, поскольку приходится запускать сборку и тестирование всех микросервисов, что будет занимать больше времени. Главным недостатком является – это знание о том, какие артефакты должны быть развернуты, а какие – нет.

Второй способ предполагает использование отдельных хранилищ кода для каждого микросервиса. Данный подход позволяет быстро вносить изме-

нение и проверять его работоспособность еще до развертывания в производственной среде. Здесь у каждого микросервиса имеется собственное хранилище исходного кода, отображаемое на его собственную CI-сборку. При внесении изменения запускается и тестируется только нужная сборка и получается один артефакт для сборки.

В качестве инструмента непрерывной поставки в данной работе используется Jenkins - программная система с открытым исходным кодом на Java, предназначенная для обеспечения процесса непрерывной интеграции программного обеспечения. Jenkins представляет собой автономный сервер автоматизации.

Jenkins позволяет описывать задачу в виде конвейера описанного либо с помощью графического интерфейса или с помощью специального Domain Specific Language называемого Jenkins Pipeline или просто Pipeline, основанный на языке программирования Groovy, который может быть представлен в декларативном или скриптовом синтаксисом. Декларативный используется чаще, так как предоставляет более богатые синтаксические функции по сравнению со скриптовым синтаксисом, и создан для того, чтобы чтение конвейера было легче. Но во многих синтаксических компонентах, таких как “scripts”, декларативного конвейера можно использовать компоненты и синтаксис скриптового Pipyline, что делает декларативный конвейер более гибким инструментом.

Также основным плюсом Jenkins являются большое количество плагинов, которые могут дополнять Jenkins широким спектром возможностей.

Jenkins Pipeline представляет собой файл, в котором описаны, с помощью специальных конструкций, стадии конвейера непрерывной поставки. Pipeline состоит из следующих блоков:

— Pipeline определяет блок, содержащий все содержимое и инструкции для выполнения всего конвейера.

— Agent указывает, где весь конвейер или определенный stage будет выполняться в среде Jenkins в зависимости от того, где находится блок agent. Agent должен быть определен на верхнем уровне внутри блока Pipeline, но использование на уровне stage необязательно.

— Блок stage обязательно должен содержать блок steps, работа всего Pipeline будет обернута в одну или более stage блоков.

— В блоке steps описываются действия которые должны быть выполнены на данной стадии Pipeline.
— Блок sh – это шаг конвейера, предоставляемый плагином Pipeline: Nodes and Processes, который выполняет данную команду оболочки.

Второй раздел «Контейнеризация» посвящен сравнению способов доставки и разворачивания программного обеспечения. В качестве основного способа выбрана контейнеризация на основе Docker, принципы функционирования которого также приводятся в этой главе.

Первый способ – использование систем управления конфигурациями. В данном способе пишутся скрипты, которые настраивают окружение для работы приложения, и поставляют само приложение. Поддержка больших конфигураций, основанных на манифестах это сложно. Изменения одного пакета, могут повлечь за собой изменения других, что может повлечь за собой множество изменений, которые могут разрушить всю систему.

Второй способ основан на парадигме неизменяемой инфраструктуры, её суть заключается в том, что, если что-то необходимо обновить, исправить или изменить, создаются новые серверы с соответствующими изменениями, которые заменяют старые серверы. Для этих целей могут использоваться виртуальные машины: создается образ, в котором зафиксированы все нужные зависимости. И разворачивать в таком случае нужно будет образ виртуальной машины с уже настроенными зависимостями для приложения. Но использование виртуальных машин имеет свои недостатки: использование виртуальных машин для изоляции приложений, а не ресурсов приводит к неэффективному использованию этих ресурсов, накладные расходы на загрузку, распространение и запуск большого образа виртуальной машины.

В качестве альтернативы виртуальным машинам в парадигме неизменяемого развертывания могут использоваться контейнеры. Контейнеры - это встроенный в ядро операционной системы Linux механизм, который позволяет запускать сразу несколько изолированных экземпляров операционных систем внутри основной операционной системы. Таким образом внутри контейнера можно создать необходимую инфраструктуру для работы приложения, а так как контейнеры встроены в механизм операционной системы и используют ядро основной операционной системы, они легковеснее, чем виртуальные машины.

В данной работе используется Docker в качестве контейнера для запуска приложений, а также в качестве артефакта. Основными компонентами Docker являются образы и контейнеры. Образ – это неизменяемый стек с кодом, библиотеками, конфигурациями и всем, что нужно для работы приложения. Контейнер – это запускаемый экземпляр образа.

Таким образом Docker-контейнеры идеально подходят для нужд доставки и разворачивания приложения и конфигурации окружения.

Третий раздел «Оркестровка» посвящен основным абстракциям платформы Kubernetes, которая является де-факто стандартом для оркестрации Docker-контейнеров.

Kubernetes – это платформа для управления контейнерами приложений, расположенных на нескольких хостах.

Платформа предоставляет множество функций управления для контейнер-ориентированных приложений, таких как автомасштабирование, развертывание без простоев, управление вычислительными ресурсами и томами. Kubernetes решает множество потребностей эксплуатации контейнеров:

— каскадное разворачивание контейнеров;
— монтирование постоянных хранилищ;
— мониторинг состояния контейнеров;
— управление ресурсами кластера;
— автомасштабирование.

Kubernetes предоставляет удобные абстракции для разворачивания и управления контейнерами:

— Pod – это наименьшая разворачиваемая часть в Kubernetes. Pod могут содержать один или несколько контейнеров. Контейнеры в одном pod работают в одном разделяемом контексте, на одном узле в разделяемом сетевом namespace и с разделяемыми volume.
— Pod не могут восстанавливаться сами по себе, когда они сталкиваются с какой-либо ошибкой. ReplicaSet гарантирует определенное количество работающих реплик пода. Когда pod будет падать, ReplicaSet будут запрашивать запуск нового пода.
— Deployment предоставляет удобную абстракцию, которая решает проблему разворачивания и откатов без простоя. Deployment поддерживает удобное каскадное разворачивание, обновление и откаты pod их реплик.

— Service – это абстрактный слой Kubernetes который определяет логический набор подов и политику доступа к ним.
— Secrets - это объекты которые хранят конфиденциальные данные в формате ключ-значения для предоставления этой информации подам, в секретах могут храниться: токены, пароли или ключи доступа.
— ConfigMap предоставляют возможность конфигурации вне Docker-образа. Это подставляет данные конфигурации в виде ключ-значение в под. В них желательно хранить не конфиденциальные данные.
— PersistentVolume – это подсистема предоставляющая API для пользователей и администраторов, которая абстрагирует детали хранения данных. Может использоваться как постоянное хранилище для баз данных и других приложений.

Данные абстракции называются ресурсами в контексте Kubernetes, они описываются в YAML формате или в формате JSON. Но чаще всего используется формат YAML поскольку он более лаконичный и удобный в использовании

Четвертый раздел «Практическая часть» описывается реализация конвейера непрерывной поставки микросервиса с выделенной базой данных в среды тестирования, разработки и в производственную среду.

В случае разработки микросервиса с выделенной базой данных, необходимо, чтобы обновления в микросервисе и базе данных поставлялись одновременно. Для достижения этого обновление базы данных включается в процесс непрерывной поставки микросервиса, хорошая практика – чтобы весь, код, который отвечает за создание и настройку базы данных хранился в одном репозитории с микросервисом. Поскольку необходимо синхронизированное состояние базы данных и микросервиса, необходимо, чтобы в случае неудачного разворачивания микросервиса происходил откат обновлений базы данных к предыдущей версии, для этого нужно чтобы на каждый скрипт, который производит изменения в базе данных, был скрипт, который бы отменял эти изменения, такие скрипты называются rollback. Поэтому к разработчику выдвигается требование: если написал, код меняющий взаимодействие с базой данных, то нужно написать соответствующий скрипт для базы данных, который можно отменить с помощью выполнения rollback.

Для сборки проекта используется система автоматической сборки Gradle.

Gradle для описания сборки и используемых зависимостей использует Groovy DSL.

Для обновления базы данных в работе используется Liquibase – независимая от базы данных библиотека с открытым исходным кодом для отслеживания, управления и применения изменений схемы базы данных.

Liquibase оперирует такими понятиями как ChangeSet – набор изменений которые должны быть применены к базе данных. В Liquibase есть несколько способов описания ChangeSet: XML-формат, JSON-формат, SQL-формат. Так как формат SQL лаконичнее и прозрачнее для разработчиков, так как им не нужно учить и запоминать еще один язык манипулирования данными, этот формат был выбран для текущего проекта.

Для непрерывной поставки обновлений было разработано два Pipeline: один для разворачивания в dev-среду, второй для разворачивания в test-среду, в которой проводятся тесты, после успешного завершения которых, происходит разворачивание в prod-среду. Первый Pipeline запускается автоматически после каждого коммита в dev-ветку репозитория на GitHub. Второй Pipeline запускается автоматически по коммиту в ветку master.

Первый Pipeline состоит из пяти стадий:

1. сборка и компиляция микросервиса;
2. сборка Docker-образа;
3. отправка Docker-образа в реестр образов DockerHub;
4. обновление базы данных;
5. обновление микросервиса в Kubernetes.

Для установки в производственную среду используется второй Pipeline, автоматически запускаемый по коммиту в систему контроля версий в ветку master.

Стадии Pipeline для test и prod-сред:

1. Сборка и компиляция микросервиса;
2. Сборка Docker-образа;
3. Отправка Docker-образа в реестр образов DockerHub;
4. Обновление базы данных в test-среде;
5. Обновление микросервиса в test-среды.
6. Запуск тестов.
7. Обновление база данных и микросервиса в prod-среде.

Пятый раздел «Демонстрация» посвящен демонстрации работы конвейера непрерывной поставки для микросервиса с выделенной базой данных. В разделе приведены удачные и неудачные выполнения конвейера поставки.

ЗАКЛЮЧЕНИЕ

В ходе данной дипломной работы была изучена специальная литература и официальная документация, которые помогли в решении поставленных задач. Была разработана система непрерывной поставки обновлений для микросервиса с выделенной базой данных. Были решены следующие задачи:

— Изучены технологии непрерывной интеграции и непрерывного разворачивания программного обеспечения, технологии контейнеризации и оркестровки контейнеров.

— Разработан конвейер непрерывной поставки обновлений для микросервиса с выделенной базой данных.

Все задачи решены, поставленная цель достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Hideto Saito, Hui-Chuan Chloe Lee, Cheng-Yang Wu. DevOps with Kubernetes: Packt Publishing Ltd, 2017. - 373.

2 Сэм Ньюмен. Создание микросервисов: Питер, 2016. — 304 с.

3 Docker in Action. Jeff Nickoloff: Manning Publications Co, 2016.- 306 с.

4 Chris Richardson. Microservices Patterns with exmaples in Java: Manning Publications Co, 2019. - 522 с.

5 Jenkins Documentation [Электронный ресурс] URL: https://jenkins.io/doc/ (Дата обращения 15.05.2019) Загл. с экрана. Яз. англ.

6 Kubernetes Documentation [Электронный ресурс] URL: https://kubernetes.io/docs/home/ (Дата обращения 15.05.2019) Загл. с экрана. Яз. англ.

7 Microservices [Электронный ресурс] URL: https://martinfowler.com/articles/microservices.html (Дата обращения 4.03.2019) Загл. с экрана. Яз. англ.

8 Pattern: Database per service [Электронный ресурс] URL: https://microservices.io/patterns/data/database-per-service.html (Дата обращения 1.05.2019) Загл. с экрана. Яз. англ.