МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ИНФРАСТРУКТУРЫ И CI/CD ЦИКЛОВ ДЛЯ СЕРВИСА СТРАХОВАНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы направления
09.03.04 — Программная инженерия
факультета КНиИТ
Королев Никита Владимирович

Научный руководитель
д.ф.-м.н., проф. _____ В.А.Романов

Заведующий кафедрой
к.ф.-м.н. _____ С.В.Миронов

Саратов 2022

ВВЕДЕНИЕ

Акутальность темы.

Архитектура программного обеспечения является одной из фундаментальной составляющей структуры программной системы и подразумевает принятие фундаментальных решений, изменение которых представляется невозможным после их реализации. Архитектурный выбор программного обеспечения имеет определенные структурные варианты проектирования, зависящих от необходимых запросов. Таким образом, требуется выбрать подходящие программные инструменты и методы реализации.

Архитектурные шаблоны представляют собой список паттернов, часто используемых решений проблем в архитектуре программного обеспечения с определенным контекстом. Одним из таких шаблонов является традиционная унифицированная модель разработки программного обеспечения монолитная архитектура. Монолитное программное обеспечение спроектировано так, чтобы быть автономным, при этом компоненты или функции программы, в отличии от модульных систем, связаны тесно. В монолитной архитектуре каждый отдельный атрибут системы и связанные с ним компоненты должны присутствовать в одном месте для выполнения кода и для запуска программного обеспечения.

Монолитные приложения являются одноуровневыми, что означает объединение нескольких компонентов в одно большое приложение. Следовательно, они, как правило, имеют большие кодовые базы, управлять которыми со временем может быть обременительно. Также, если необходимо обновить один компонент программы, другие элементы также могут потребовать перезаписи, и все приложение должно быть перекомпилировано и протестировано. Этот процесс может занять много времени и может ограничить гибкость и скорость групп разработчиков программного обеспечения. Вдобавок, по мере роста

пользователей возникнет проблема, когда система не сможет обрабатывать столько запросов к монолитному приложению и потребуется масштабируемость, эластичность и отказоустойчивость.

Одним из подходов является отделить часто используемый модуль внутри монолита, таким образом, получая распределенный монолит. В архитектуре распределенного монолита удаляются все основные части из основного монолита и в конечном итоге запускаются независимые сервисы. По определению, распределенный монолит — это приложение, развернутое как микросервис, но построенное как монолит.

Одним из недостатков этого типа архитектуры является то, что все компоненты должны быть развернуты или выпущены одновременно, поскольку база данных используется совместно этими службами. Кроме того, чтобы обойти обслуживание внешнего интерфейса, в такой системе был введен API-шлюз. В этом случае внешнее веб-приложение концентрируется только на одной конечной точке, где шлюз API знает о маршрутизации этих конкретных запросов API в правильные службы. Все приложение объединено в единый пакет с монолитной архитектурой. С распределенным монолитом проблемы тяжести и негибкости монолитов осталась актуальна.

Данные проблемы потребовали других решений и основное внимание в данной работе уделено архитектуре микросервисов.

Архитектура микросервисов — вариант структурного стиля сервис-ориентированной архитектуры (SOA) — представляет приложение, как набор слабо связанных сервисов. Цель состоит в том, чтобы команды могли реализовывать свои изменения независимо от других. Слабая связность уменьшает все типы зависимостей и порождаемые этим сложности, поскольку разработчикам не нужно заботиться о времени недееспособности сервиса для пользователей и необходимости перекомпилировать приложение. Интерфейсы должны быть тщательно спроектированы и рассматриваться как общедоступный

API. Одним из используемых методов является наличие нескольких интерфейсов в одной и той же службе или нескольких версий одной и той же службы, чтобы не нарушать работу существующих пользователей кода [1].

Целью настоящей работы является разработка инфраструктуры и CI/CD циклов для сервиса страхования, который представляет собой клиент-серверное приложение, реализованное на языках программирования Java и JavaScript с использованием фреймворков, готовых структур, на основе которых можно создавать программное обеспечение. Заказчиком данного проекта является руководство «Манго Иншуринг» у компании «Неофлекс Консалтинг». Руководством «Манго Иншуринг» был определен список используемых технологий (Nomad, Consul, Concourse CI, Terraform), данные технологии имеют открытый исходный код, позволяя экономить на эксплуатации, также обозначен облачный провайдер, предоставляющий услуги VK Cloud Solutions и поставлены следующие задачи:

- исследовать утвержденныете хнологии (Nomad, Consul, ConcourseCI, Terraform) и их аналоги;
- разработать сценарии создания и настройки конфигураций виртуальных машин для микросервисов продукта;
- настроить сетевое взаимодействие виртуальных машин;
- реализовать отказоустойчивость микросервисов продукта;
- разработать сценарии тиражирования и обновления микросервисов на виртуальных машинах.

Структура и объём работы.

Бакалаврская работа состоит из введения, 3 глав, заключения, списка использованных источников и 3 приложений. Общий объем работы – 71 страница, из них 39 страниц — основное содержание, включая 16 рисунков, список использованных источников информации — 10 наименований.

1 Создание сценариев управления облачными ресурсами

1.1 Облачные технологии

Основной технологией, обеспечивающей облачные вычисления, является виртуализация. Программное обеспечение виртуализации разделяет физическое вычислительное устройство на одно или несколько «виртуальных» устройств, каждое из которых можно легко использовать и управлять им для выполнения вычислительных задач. С виртуализацией на уровне операционной системы, по существу, создавая масштабируемую систему из нескольких независимых вычислительных устройств, простаивающие вычислительные ресурсы могут распределяться и использоваться более эффективно. Виртуализация обеспечивает гибкость, необходимую для ускорения IT операций, и снижает затраты за счет повышения степени использования инфраструктуры. Автономные вычисления автоматизируют процесс, посредством которого пользователь может выделять ресурсы потребованию, сводя к минимуму участие пользователя, автоматизация ускоряет процесс, снижает трудозатраты и снижает вероятность человеческих ошибок.

1.2 Инструмент управления инфраструктурой

Попутно с развитием облачных технологии, зарождались новые программные продукты. Terraform стал одним их программных решений позволяющих выделять облачные ресурсы и управлять ими с помощью нескольких ключевых операций, этот инструмент позволяет определять как облачные, так и локальные ресурсы в удобочитаемых файлах конфигурации, которые можно версионировать, повторно использовать и делиться ими. Также есть возможность использовать согласованный рабочий процесс для подготовки и управления всей инфраструктурой на протяжении всего ее жизненного цикла. Terraform может управлять компонентами низкого уровня, такими как вычислительные ресурсы, хранилище и сетевые ресурсы, а также компонентами высокого уровня, такими как записи DNS.

Terraform создает и управляет ресурсами на облачных платформах и других сервисах через свой API. Провайдеры позволяют

Terraform работать практически с любой платформой или сервисом с помощью доступного API.

1.3 Инструмент оркестрации

В связи с ростом количества разворачиваемых компонентов системе становится все труднее управлять ими всеми. Google, вероятно, была первой компанией, которая поняла, что в такой ситуации необходим гораздо более оптимальный способ развертывания и управления программными компонентами и инфраструктурой, с тем чтобы обеспечить масштабируемость в глобальном масштабе.

В 2014 году Google анонсировала Kubernetes, систему с открытым исходным кодом. Kubernetes — это система оркестровки контейнеров, которая теперь управляется Cloud Native Computing Foundation (CNCF) и разрабатывается Google, Red Hat и многими другими. Kubernetes и Nomad поддерживают схожие основные варианты использования для тиражирования приложений и управления ими, но они отличаются несколькими ключевыми моментами. Kubernetes стремится предоставить все функции, необходимые для запуска приложений на основе контейнеров Linux, включая управление кластером, планирование, обнаружение сервисов, мониторинг, управление секретами и многое другое. Nomad стремится сосредоточиться только на управлении кластерами и разработан в соответствии с философией Unix, заключающейся в том, чтобы иметь небольшую область действия при составлении, достигая этого, с помощью таких инструментов, как Consul для обнаружения сервисов или сервисной сетки.

Nomad архитектурно намного проще Kubernetes и представляет собой единый бинарный файл как для клиентов, так и для серверов, не требуя внешних служб для координации или хранения. Nomad сочетает в себе легкий менеджер ресурсов и сложный планировщик в одной системе.

1.4 Consul и его роль в продукте

Consul — это сервисная сетка, предоставляющая полнофункциональную плоскость управления с функциями

обнаружения, настройки и сегментации сервисов. Каждую из этих функций можно использовать по отдельности по мере необходимости, или их можно использовать вместе для создания полной сервисной сетки. Консулу требуется плоскость данных, и он поддерживает как прокси, так и собственную модель интеграции. Consul решает проблемы, с которыми сталкиваются организации любого размера с архитектурой микросервисов. Это варьируется от работы в различных распределенных средах и географических местоположениях до удовлетворения потребности в защите всего трафика приложений. Consul позволяет организациям использовать модель нулевого доверия при масштабировании.

1.5 Инструмент настройки CI

В работе используется инструмент настройки CI —Concourse. Это система, построенная со слабо связанной архитектурой микросервисов; узел базы данных, использующий PostgreSQL, является единственным состоянием, сохраняемым системой, где хранятся все истории сборки, сохраненные пайплайны, а также привилегии доступа пользователей и групп.

Служба веб-узла предоставляет пользовательский интерфейс для Concourse CI, с помощью которого разработчики и администраторы могут быстро просмотреть свои пайплайны, в том числе их статус. Сломанные конвейеры можно легко идентифицировать, чтобы пользователи могли исправить любые проблемы.

Рабочие узлы выполняют каждую из задач, определенных в конвейере Concourse CI. Они загружают образы контейнеров, клонируют репозитории git и запускают тесты, как определено.

Конвейеры Concourse CI строятся с использованием трех разных парадигм абстракции: задач, заданий и ресурсов.

Задачи — это наименьшая единица работы, которую выполняет Concourse CI. Их можно вызвать для запуска скрипта или даже одной команды в тестовом контейнере. При этом выходные данные задачи предоставляются в виде подробных файлов журналов, которые при необходимости можно анализировать программно.

Работа представляет собой пакет задач. Объединив группу задач в задание, пользователи Concourse CI могут сделать свой конвейерный код пригодным для повторного использования в других системах. Такое объединение обеспечивает абстракцию более высокого уровня по мере того, как конвейеры становятся больше и сложнее, что облегчает новым членам команды освоение скорости.

Ресурсы — это то, над чем задания выполняют действия. Типичным примером может служить репозиторий git; После настройки Concourse CI может загружать новый код для тестирования, запускать тестовые сценарии, хранящиеся в git, или даже отправлять свои собственные изменения. А поскольку все в Concourse CI настроено как код, даже конфигурациями ресурсов можно управлять в git и повторно использовать в организации.

1.6 Схема инфраструктуры

Описание и формирование инфраструктуры подразумевает анализ архитектуры внутреннего взаимодействия для последовательного создания виртуальных машин и тиражирования программных продуктов.

Ниже перечислены ключевые элементы системы.

Менеджер репозиториев Nexus Sonatype, который организует, хранит и распределяет артефакты, необходимые для разработки.

Главная, руководящая процессами балансировки нагрузки виртуальная машина с предустановленными Nomad и Consul, а также несколько управляемых серверов, на которых запускаются сервисы приложения.

Инструмент настройки инфраструктуры Terraform, через который осуществляется управление ресурсами облака.

CI сервер Concourse, обеспечивающий сборку и тиражирование образов приложения на необходимые виртуальные машины, а также обратный HTTP-прокси Traefik для направления запросов из внешней сети к компонентам внутренней. Traefik интегрируется с существующими компонентами инфраструктуры и настраивается автоматически и динамически. Он поставляется с мощным набором промежуточного программного обеспечения, которое расширяет его

возможности, включая балансировку нагрузки, шлюз API, вход Orchestrator, а также связь между службами восток-запад и многое другое [3].

Общий процесс функционирования и обновления системы начинается с разработки кода. Затем, данный код собирается для backend и frontend с помощью Apache Maven, инструмента для управления и понимания программных проектов, который основываясь на концепции объектной модели (POM), способен управлять сборкой проекта, и менеджера пакетов для языка программирования JavaScript, Npm, соответственно. Собранный код отправляется в удаленный репозиторий Bitbucket, после чего, в момент запуска сборки Concourse CI, помещаясь в среду исполнения (контейнер), обновляет ранее развернутый сервис или же тиражируется новым микросервисом. Процесс контейнеризации осуществляется благодаря полученным программным компонентам из менеджера репозиториев Nexus Sonatype. В свою очередь процессом замещения или тиражирования на виртуальных машинах исполнителях управляет Nomad (руководящий сервер).

Исходя из архитектуры приложения и анализа взаимодействия был разработан сценарий, включающий в себя следующие шаги:

1. Создание, виртуальных машин на облачном провайдере;
2. установка Nomad и произведение конфигурационных настроек;
3. добавление и настройка Sonatype Nexus;
4. установка Consul и добавление необходимых модулей;
5. проверка работы.

1.7 Тиражирование Nomad и Consul

На примере сценариев создания виртуальных машин и тиражирования Consul, Nomad, показаны основные возможности решения задач по формированию всей инфраструктуры продукта [4].

1.8 Настройка сетевого взаимодействия

Обеспечение безопасности в компьютерных сетях — это основное условие защиты конфиденциальных данных от разного рода

угроз, таких как шпионаж, уничтожение файлов и прочие несанкционированные действия [5].

Одной из распространенных сетевых угроз является несанкционированный доступ извне, причем не только умышленный, но и случайный [6]. Таким образом общая задача заключает в том, чтобы локализовать нашу сеть, оставляя лишь необходимые выходы в глобальную сеть [7].

Для осуществления данной задумки требуется настроить корпоративный VPN, с помощью которого можно будет взаимодействовать с сервисами локальной сети.

OpenVPN — это система виртуальной частной сети (VPN), которая реализует методы создания безопасных соединений «точка-точка» или «сеть-сеть» в маршрутизируемых или мостовых конфигурациях и средствах удаленного доступа. Он реализует как клиентские, так и серверные приложения, позволяет одноранговым узлам аутентифицировать друг друга, используя предварительно общие секретные ключи, сертификаты или имя пользователя с паролем.

2 Настройка виртуальных машин исполнителей и CI сервера

2.1 Реализация сценариев настройки конфигурации для виртуальных машин исполнителей и CI сервера

Требовалось развернуть сервер сборки CI/CD Concourse, с помощью которого затем можно было бы тиражировать программное обеспечение, и рабочие виртуальные машины, которые используются для запуска контейнерных приложений и управления сетью, чтобы обеспечить надлежащее облегчение трафика между приложениями внутри кластера и извне [9]. А также создать и тиражировать Nexus Sonatype, где будут храниться все образы сервисов и зависимости сборщиков пакетов приложения.

После создания виртуальных машин требуется произвести установку и настройку конфигураций каждого отдельной виртуальной машины. Устанавливается необходимый список программных компонентов, сформированный исходя из зависимостей технологии, а также прописываются необходимые IP адреса для DNS конфигураций.

После вышеперечисленных шагов инфраструктура готова и можно размещать микросервисы приложения.

3 Реализация CI/CD цикла

3.1 CI/CD цикл

CI/CD — это метод частой доставки приложений клиентам путем внедрения автоматизации на этапах разработки приложений. Основными концепциями CI/CD являются непрерывная интеграция, непрерывная поставка и непрерывное развертывание. CI/CD — это решение проблем, которые интеграция нового кода может вызвать для групп разработки и эксплуатации. Многие предприятия начинают с добавления CI, а затем постепенно продвигаются к автоматизации доставки и развертывания, например, в рамках облачных приложений.

3.1.1 Continuous integration

В современной разработке приложений цель состоит в том, чтобы несколько разработчиков одновременно работали над разными функциями одного и того же приложения.

Непрерывная интеграция (CI) помогает разработчикам объединять изменения кода обратно в общую ветвь даже каждый день. После слияния изменений эти изменения проверяются путем автоматического создания приложения и выполнения различных уровней автоматизированного тестирования, обычно модульных и интеграционных тестов, чтобы убедиться, что изменения не нарушили работу приложения. Если автоматизированное тестирование обнаруживает конфликт между новым и существующим кодом, CI облегчает быстрое и частое исправление этих ошибок.

3.1.2 Continuous delivery

После автоматизации сборок, модульного и интеграционного тестирования в CI непрерывная поставка автоматизирует выпуск проверенного кода в репозиторий. Итак, чтобы иметь эффективный процесс непрерывной доставки, важно, чтобы CI уже был встроен в конвейер разработки. Цель непрерывной доставки — иметь кодовую базу, всегда готовую к развертыванию в производственной среде.

3.1.3 Continuous deployment

Заключительный этап зрелого конвейера CI/CD — непрерывное развертывание. В качестве расширения непрерывной доставки, которое автоматизирует выпуск готовой к работе сборки в репозиторий кода, непрерывное развертывание автоматизирует выпуск приложения в рабочую среду. Поскольку на этапе конвейера перед производством нет ручного шлюза, непрерывное развертывание в значительной степени зависит от хорошо продуманной автоматизации тестирования.

3.2 Реализация сценариев тиражирования сервисов

Выполнение сценариев тиражирования осуществляется с помощью инструмента сборок Concourse CI [10]. Используются шаблоны инструкций тиражирования, которые запускаются по нажатию сборки. Необходимые сервисы вынесены в отдельную инструкцию, которая передает параметры в шаблон из которого реализуется сборка и тиражирование.

ЗАКЛЮЧЕНИЕ

Целью работы являлась разработка инфраструктуры и CI/CD циклов для сервиса страхования.

Были выполнены следующие задачи:

— исследованы утвержденные технологии (Nomad, Consul, Concourse CI,

Terraform) и их аналоги;

— разработаны сценарии создания и настройки конфигураций виртуальных машин;

— произведена настройка сетевого взаимодействия виртуальных машин;

— реализована отказоустойчивость микросервисов продукта;

— разработаны сценарии тиражирования сервисов и обновления микросервисов на виртуальных машинах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Дженнифер, Д.* Философия DevOps. Искусство управления IT / Д. Дженнифер. — Питер, 2017.

2 *sabharwal, N.* Infrastructure-as-Code Automation Using Terraform, Packer, Vault, Nomad and Consul: Hands-on Deployment, Configuration, and Best Practices / N. sabharwal. — Springer Nature B.V., 2021.

3 Terraform[Электронныйресурс]. — URL: https://www.techtarget.com/searchitoperations/ (Дата обращения 23.04.2022) Загл. с экр. Яз. англ.

4 *Брикман, Е.* Terraform. Инфраструктура на уровне кода / Е. Брикман. — Питер, 2020.

5 *Bird, J.* DevOps for Finance / J. Bird. — O'Reilly Media, Inc., 2015.

6 What is a ci/cd pipeline [Электронный ресурс]. — URL: https://www.redhat.com/en/topics/devops/what-cicd-pipeline (Дата обращения 21.04.2022) Загл. с экр. Яз. англ.

7 *Тейнсли, Д.* Linux и UNIX: программирование в shell. Руководство разработчика / Д. Тейнсли. — Издательская группа BHV, 2001.

8 Shell scripting for beginners ? how to write bash scripts in linux [Электронный ресурс]. — URL: https://www.freecodecamp.org/news/shell-scripting-crash-course-how-towrite-bash-scripts-in-linux/ (Дата обращения 20.04.2022) Загл. с экр. Яз. англ.

9 *Jackson,R.*SimplifyingServiceManagementwithConsul:Overcomeconnectivity and security challenges within dynamic service architectures / R. Jackson. — Packt Publishing, 2021.

10 Concourse ci docs [Электронный ресурс]. — URL: https://concourseci.org/docs.html (Дата обращения 19.04.2022) Загл. с экр. Яз. англ.