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

СОЗДАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ СЕТЕВОЙ ИНФРАСТРУКТУРОЙ УНИВЕРСИТЕТА

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 5 курса 551 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Филиппова Ивана Александровича

Научный руководитель
к. т.н., доцент _____ В. М. Соловьёв

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

Саратов 2024

ВВЕДЕНИЕ

Актуальность темы. Сетевая инфраструктура университета является основополагающим элементом, поддерживающим академические, административные и исследовательские операции. Сложный аппаратно-программный комплекс, состоящий из инженерных сетей, оборудования коммутации и маршрутизации и системы управления, который требует постоянного контроля и обслуживания. Для эффективного управления этим комплексом необходимо иметь надежную систему, которая позволит быстро реагировать на любые проблемы и решать их своевременно.

В современную эпоху быстрой цифровизации и растущей зависимости от сетевых сервисов, университеты сталкиваются с необходимостью находить инновационные решения для оптимизации управления своей сетевой инфраструктурой. Традиционные методы управления больше не соответствуют требованиям масштабируемости, автоматизации и обеспечения безопасности, что приводит к снижению эффективности, увеличению затрат, потенциальным рискам для безопасности и не обеспечивают необходимую скорость реакции специалистов на возникающие проблемы.

Цель данной работы является разработка и внедрение системы управления сетевой инфраструктурой университета, которая будет включать в себя веб-приложение для администратора, автоматическую систему настройки и управления сетевой конфигурацией, систему мониторинга инфраструктуры, а также систему создания конфигурационных файлов для Nginx. Кроме того, планируется ввести общий стандарт разработки, настройки и доставки систем автоматизации.

Постановленная цель определила **следующие задачи:**

1. Разработать, настроить и запустить в промышленной среде веб-приложение для управления администратором. Данное приложение должно использовать существующую базу данных клиентов, которая была исторически заполнена, умела оперировать с этими данными, удалять, добавлять или изменять их, а так же могла средствами API вызывать другие системы для их управления.

2. Разработать приложения для автоматической системы настройки и управления сетевой конфигурацией пользователей (хостов) на основе базы данных. используемой выше.
3. Разработать систему динамического мониторинга инфраструктуры.
4. Разработать систему создания конфигурационных файлов сервиса Nginx для динамического изменения логики работы сетевого взаимодействия с сайтами университета.
5. Ввести общий стандарт разработки, настройки и доставки будущих систем автоматизации.

Для написания теоретической части работы, использовалась учебная и монографическая литература.

Данная работа состоит из введения, трех глав, заключения, списка литературы и приложения.

1 Основное содержание работы

Введение описывает общую концепцию работы системы управления сетевой инфраструктурой университета. Описывается, что система будет централизованно контролировать и оптимизировать все изменения, производимые на сетевой инфраструктуре, включая маршрутизацию, коммутацию, безопасность и управление производительностью. Также подчеркивается, что система позволит университету улучшить свою сетевую среду, обеспечить надежный доступ к критически важным ресурсам и услугам, оптимизировать использование ресурсов и существенно уменьшить риск возникновения ошибки конфигурации.

Первая глава посвящена обзору сетевой инфраструктуры университета и задачам, которые решают сетевые администраторы. Описывается необходимость создания системы управления сетевой инфраструктурой и её преимущества. Также приводится историческая справка о создании и развитии отдела УЦИТ (ПРЦНИТ) при Саратовском Государственном Университете.

Раздел 1.1 описывает все недостатки существующих систем и подходов, выделяя критические моменты и недостатки. Объясняется основная задача и актуальность перехода от системы распределенного управления к микросервисной архитектуре

В разделе 1.2 выделяются ключевые места будущей разработки, такие как:

- DHCP и DNS
- Хранилища информации пользователей
- Хостинговая система сайтов
- Почтовая система
- Телефония
- Система управления пользователями

а так же основные свойства и требования к новой архитектуре:

- Сохранение текущей базы данных клиентов
- Система мониторинга
- Автогенерация конфигураций
- Удаленное управление
- Простота использования и обновления
- Безопасность
- Низкий порог вхождения

- Открытый исходный код и низкая стоимость продукта

Формируется более детальный список задач для решения цели работы. Формируется список стандартов и критериев разрабатываемого решения:

- Каждый новый или существующий компонент должен быть выполнен на одном и том же языке программирования с соблюдением требований безопасности, чистоты кода и имеющий возможность работать независимо от других программ.
- Компоненты должны уметь работать на разных операционных системах, иметь обратную совместимость, простоту реализации и открытый исходный код для сотрудников.
- Один компонент должен обслуживать только одну бизнес логику функционала для которого он создается, не заходя в смежные системы.
- Компоненты должны просто собираться, интегрироваться, настраиваться и эксплуатироваться.
- Должен быть реализован внутренний функционал для мониторинга данной системы.

Глава 2 раскрывает анализ существующих языков программирования. Проводится сравнительный анализ для выявления плюсов и минусов при выборе языка разработки. Определяются основные инструменты и их краткое описание и причины выбора. Так же задается основные критерии выбора языка программирования:

- Язык должен быть компилируемый
- Язык должен быть строго типизированный
- Постоянно обновляемый
- Отсутствие низкоуровневого управления памятью
- Простое управления параллелизмом
- Язык имеет большое сообщество пользователей
- Не слишком сложный язык для изучения
- Большая стандартная библиотека или множество пользовательских

Раздел 2.1 описывает язык программирования Python. Язык позволяет разработчикам создавать сложные и масштабируемые системы, используя модульность и объектно-ориентированный подход. Python также хорошо подходит для создания API и взаимодействия между различными компонентами системы.

Python является де-факто в мире программирования, как один из самых старых языков. Уровень вхождения в него очень низкий и любой разработчик сможет с первых минут начать его использовать. Гигантское сообщество, множество готовых решений и постоянные обновления языка, делают его одним из самых удобных языков для разработки. Но ряд минусов, такие как типизация и интерпретация, делают его не достаточно пригодным для решения поставленных задач. Так что данный язык остается хорошей альтернативой в тех случая, когда невозможно решить задачу другими языками.

Раздел 2.2 описывает анализ языка Golang. Данный язык идеально подходит под все требования и закрывает ряд задач. У него есть ряд минусов, но они являются не существенными для решения текущих задач, а скорее даже плюсами. Исходя из всего вышесказанного, данный язык будет основным языком для разработки. На нем будет очень удобно писать различное программное обеспечение, но для создания очень большой и сложной бизнес логики он подходит не полностью, а все из за отсутствия простой библиотеки для работы с базами данных и удобного способа создания веб-приложений. Таким образом, данный язык остается основным для написания простого, понятно и быстро выполняемого кода, но требует дальнейшего анализа остальных языков программирования для выбора более оптимального решения для разработки более сложных бизнес-решений.

Раздел 2.3 посвящен анализу языка Scala. Данный язык полностью покрывает все установленные требования. С его помощью можно будет легко и просто создавать огромные системы и программы. Хотя язык и является достаточно сложным для изучения новичкам, но порог вхождения в него не является настолько высоким, как например. в C++ или Asembler. Так что данный язык будет использоваться для создания более сложных систем.

В разделе 2.4 проводится анализ и выбор готовых инструментов автоматизации, сокращающих количество действий для вывода разрабатываемого продукта до конечного пользоваться или услуги. О данных инструментах, зачем они необходимы и как будут использоваться будет рассказано в последующих главах. В ряды таких инструментов будут входить:

- Bitbucket - основное хранилище исходного кода

- Jenkins - как оркестратор и основной инструмент для автоматической сборки приложений
- Nexus - Система для хранения собранных артефактов и последующего распространения
- SonarQube - Система проверки кода на ошибки безопасности
- LDAP - MS Active Directory в роли каталога пользователей и устройств, а так же системой управления и разграничений прав
- Базы данных - PostgresQL и ClickHouse для хранения пользователей и логов
- Prometheus - как система мониторинга приложений

При разработке же приложений будут использовать две базы данных. PostgresQL - как реляционная база данных для хранения персональных данных, настроек и информации о работе системы. И ClickHouse - колоночная аналитическая база, созданная для быстрой обработки больших объемов структурированных данных. Данная база данных будет использоваться для хранения и анализа статистики и логов приложений. Если в каком либо приложении нам необходимо реализовать схему разделение по ролям или пользователям, то необходимо использовать уже готовую и заполненную систему LDAP, заранее заполненную ролям и группам.

Для полноценной работы компонентов недостаточно только написать, собрать и внедрить. Каждый компонент так же необходимо сопровождать и поддерживать. Система мониторинга является одной из самых важных частей любого IT-проекта. Если сопровождающий персонал не знает что происходит у него внутри приложения, как оно себя ведет, какая у него динамика и так далее, то своевременная реагирование на внештатную активность, сбой или просто повышенный поток клиентов, не представляется возможным. Таким образом, система на основе логирования и метрик Prometheus будет являться основным выборов как система мониторинга.

Глава 3 рассказывает о процессе разработки конечного решения данного проекта.

Система должна иметь модель микросервисной архитектуры, где каждый компонент независимо отвечает за свой участок бизнес логики и управления тем, или иным объектом всего IT-ландшафта. Каждое программное обеспече-

ние, предоставляющее какую либо IT-услугу конечным пользователям. должно быть полностью или частично быть автоматизированным для программной настройки и управления, а так же должно существовать центральная точка входа (управления) всеми данными системами. В качестве центральной точки входа будет реализовано веб-приложение, использующее базу данных для хранения той или иной конфигурации, а так же вызывающие отдельные компоненты всего IT-ландшафта для его управления и конфигурирования.

Раздел 3.1 описывает разработку панели администрирования. Разработка будет происходить с помощью языка Scala и базы данныхы PostgreSQL. При этом будет переиспользована существующая база данных, так как разработка производиться на уже годами устоявшийся IT-ландшафт. Разработка панели будет производиться по модели MVC с использованием ряда готовых библиотек. В их число входит:

- Play Framework - мощный фреймворк для разработки веб-приложений
- Slick - для работы с базой данных
- Bootstrap - Для красивого и удобного отображения визуальной части приложения
- LDAPsdk - библиотека для возможности авторизации пользователей в LDAP.
- Silhouette - библиотека для авторизации пользователей в разрабатываемой системе.

Сама разработка будет поделена на несколько частей:

- Работа с базой данных и её моделями
- Написание бизнес-логики работы и обработки данных (BackEnd часть)
- Визуальное отображение пользовательского интерфейса (FrontEnd часть)

В процессе разработки анализируется существующая база данных. Выделяются основные таблицы для разработки. Эти таблицы преобразуем в структуры данных выбранного языка программирования, которые в дальнейшем будут называться моделями. Каждый поле из данных таблиц описываем как класс на языке Scala с соответствующим типом данных.

После создания всех нужных для работы моделей, необходимо связать их с соответствующими таблицами в базе данных. Теперь в будущем, используя модель, будут вызываться данные сразу из таблицы.

Так как Play Framework работает по паттерну MVC и слой Model был уже создан, далее следует создание слоя Controller. В данном слое будет выполняться основная бизнес-логика приложения. Здесь будут реализовывать такие действия, как проверка на правильность введенных данных, работу с базой данных, вызов сторонних сервисов, алгоритмы, сортировки и так далее. В конце добавим дополнительный функционал, который при обращении по API будет возвращать Json-файл. Дальше идет разработка слоя View, который будет отображать пользовательский интерфейс.

Так как Play Framework заточен на создания веб-приложений, он имеет мощную систему шаблонизации, которую и будем использовать. Так как многие действия, например. отображение списка данных или навигационная панель. должны часто повторяться, то и данные HTML страницы будут созданы всего один раз, но вызываться во многих частях сайта.

Результат запущенного проекта можно увидеть в приложении. Там в пример можно увидеть страницу авторизации или страницу добавления нового устройства.

В разделе 3.2 описывается использование инструментов автоматизации, их настройка и использование. Для того, что бы приложением начали пользоваться, его необходимо вывести в промышленную эксплуатацию. Для этого необходимо применить ряд действий:

- Скомпилировать приложение и упаковать его исходный код в специальный пакет (этап сборки)
- Провести тестирование данного приложения на безопасность
- Написать инструкции установки и запуска данного пакета на определенной операционной системе.
- Установить данный пакет по инструкциям на операционную систему
- Запустить данные приложение

Сборка проекта необходима для того, чтобы преобразовать исходный код программы в исполняемый файл. В процессе сборки происходит компиляция исходного кода, линковка объектных файлов, создание библиотек и других необходимых ресурсов. В результате сборки получается готовый к запуску исполняемый файл или пакет, который можно установить и использовать.

Для сборки готового программного обеспечения будет использован инструмент SBT. Это пакетный менеджер языка программирования Scala. Данный менеджер скачает все зависимые библиотеки, соберет исходный код и создаст исполняемый Jar-пакет, который можно будет запустить на промышленном сервере.

После выполняется команда сборки и получается готовый RPM-пакет для установки на операционную среду. После того, как RPM-пакет был собран, его можно скопировать на выделенный сервер для установки и использования.

Для возможности устанавливать RPM-пакет в любое время и хранения истории его изменений, все собранные пакеты должны хранится в некотором хранилище артефактов. В данном случае будет выступать готовый инструмент Nexus. Это частный репозиторий, который используется для хранения артефактов программного обеспечения. Он позволяет управлять зависимостями в проекте, контролировать качество используемых библиотек и компонентов, а также оптимизировать процесс сборки приложений. Теперь данный пакет можно устанавливать на любом сервер, не тратя каждый раз время на его копирование.

Все данные этапы необходимо автоматизировать средствами CI/CD как непрерывный конвейер разработки, тестирования и развертывания. Данный конвейер будет так же применяться к любой разработке любой внутреннего сервиса университета и может спокойно масштабироваться. Для автоматизации сборки будет использоваться готовый и установленный инструмент оркестрации Jenkins.

Результатом будет полностью автоматизированный конвейер для сборки, тестирования и интеграции сервисов. Данные действия призваны для создания простой и автоматической среды разработки программного обеспечения, сокращающего время создания сервисов и их доставки на промышленные среды. Пример полностью пройденного конвейера сборки:

В разделе 3.3 описывается разработка сервиса генерации сетевой конфигурации. на основе DHCP. DHCP (Dynamic Host Configuration Protocol) - это сетевой протокол, который позволяет сетевым устройствам автоматически получать IP-адрес и другие параметры, необходимые для работы в сети TCP/IP. Этот протокол работает по модели "клиент-сервер где компьютер-клиент обращается к серверу DHCP и получает от него нужные параметры.

Данная система является основной, с который большую часть времени работаю сетевые администраторы. Внедрение автоматизации в управление данной системой - одна из главных задач любого IT-предприятия.

Сервис по API будет постоянно опрашивать панель администрирования и получать список устройство и их настройки сети. Вся данная информация будет содержаться в памяти сервиса и обновляться в момент изменения списка устройств или их свойств. При изменений данных, будут генерироваться конфигурационные файлы для сервера DHCP и вноситься изменения. После будет производиться перезагрузка сервера DHCP и новые/старые устройства будут получать обновленную информацию о сети и получать возможность входа в сеть. Схема работы изображена ниже.

Данный сервис будет написан на языке Golang, так как требует быстрого выполнения, удобного параллелизма и не будет занимать кода. Полный код можно будет увидеть здесь. Данный сервис, как и все предыдущие разработанные системы, так же автоматизируется установкой в систему непрерывного конвейера CI/CD.

В разделе 3.4 описывается разработка системы генерации конфигурационных файлов для Nginx. Nginx — это бесплатное программное обеспечение, которое используют для работы сайтов. В нём есть функции веб-сервера и почтового прокси-сервера. Принцип его разработки полностью совпадает с принципом разработки сервиса для генерации конфигурации DHCP.

Здесь так же создается подключение к API панели администрирования. которая раз в какое то время опрашивает панель на список доступных сайтов в базе данных, проверят их и добавляет в общий список сайтов. для которых необходимо создать файл конфигурации Nginx, после перезагружает сервер.

Отличительной особенностью от прошлого сервиса является наличие локального конфигурационного файла. Создано для тех случаев, когда не хочется подключать данную систему к панели администрирования и использовать её как самостоятельное программное обеспечение.

Раздел 3.5 заканчивает описание разработки создание системы генерации и настройки мониторинга на основе Prometheus. Сервис который будет генерировать данный список устройств (хостов). Данный сервис работает. опять же. по

схожему принципу. как и прошлые сервисы. Подключается к панели управления, забирает список устройств и добавляет их в систему мониторинга.

Таким образом. была показана схема реализации общей панели администрирования и примеры реализации различных сервисов управления системами IT. Зафиксирована общая документация и стандарт разработки. внедрения новых сервисов и систем. В будущем разработчики будут сами реализовывать и внедрять новый сервисы в уже существующую архитектуру. сразу автоматизируя свою работу и выигрывая себе большое количество свободного времени.

В заключении создание системы управления сетевой инфраструктурой университета является важным шагом для оптимизации работы учебного заведения. Данное решение позволяет эффективно управлять ресурсами, повышать качество образования и улучшать условия обучения студентов.

Была произведена большая работа по разработке и внедрению данной системы, которая включает в себя автоматизацию процессов управления зданиями, инженерными сетями и оборудованием. Благодаря этому, специалисты технической поддержки могут быстро реагировать на любые проблемы и решать их своевременно.

Разработанная система управления сетевой инфраструктурой университета предоставляет всеобъемлющее решение для эффективного мониторинга, управления и обслуживания объектов университета. Система объединяет передовые технологии и лучшие практики, обеспечивая следующие преимущества:

- Централизованная платформа данных обеспечивает доступ к своевременной и точной информации для принятия обоснованных решений.
- Система автоматизирует повторяющиеся и трудоемкие задачи, такие как генерация рабочих ордеров, планирование обслуживания и отслеживание инвентаря. Это освобождает персонал по обслуживанию, позволяя им сосредоточиться на более стратегических инициативах.
- Система позволяет оперативно получать информацию о состоянии инфраструктуры университета и быстро реагировать на возникающие проблемы.