**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Факультет инфокоммуникационных технологий**

**КУРСОВАЯ РАБОТА**

***По дисциплине:***

**«*Сетевое программирование*»**

**На тему:**

**«*Анализ инструментов сетевой автоматизации*»**

**Выполнила:**

студентка группы К34212

Новожилова Анна Владимировна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Проверил преподаватель:**

Филянин Иван Викторович, аспирант ФПИ и КТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отметка о выполнении:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Санкт-Петербург 2022 г.

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Студент** | Новожилова Анна Владимировна | | | | | |  |
|  |  |  |  |  |  |  | (Фамилия И.О.) |
| **Факультет** |  | Инфокоммуникационных Технологий | | | | | |
| **Группа** | К34212 | | |  |  |  |  |
| **Направление (специальность)** | | | | | | 11.03.02. – Инфокоммуникационные технологии и системы связи | |
| **Руководитель** | | Филянин Иван Викторович, аспирант ФПИ и КТ | | | | | |
| (Фамилия И.О., должность, ученое звание, степень) | | | | | | | |
| **Дисциплина** | | Сетевое программирование | | | | | |
| **Наименование темы** | | | | Анализ инструментов сетевой автоматизации | | | |
| **Задание** | Провести сравнительный анализ популярных инструментов автоматизации управления | | | | | | |
| конфигурацией в сети | | | | | | | |
|  | | | | | | | |
| **Краткие методические указания** | | | | | | | |
|  | | | | | | | |
| **Содержание пояснительной записки** | | | | | | |  |
| Пояснительная записка включает разделы - введение, определения, теоретическая часть, практическая часть | | | | | | | |
| **Рекомендуемая литература** | | | | | | | |
|  | | | | | | | |
|  | | | | | | | |
| Руководитель | | Филянин Иван Викторович | | | | | |
|  |  |  |  |  |  |  | (Подпись, дата) |
| Студент | Новожилова Анна Владимировна | | | | | |  |
|  |  |  |  |  |  |  | (Подпись, дата) |

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**ГРАФИК ВЫПОЛНЕНИЯ КУРСОВОЙ РАБОТЫ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Студент** | Новожилова Анна Владимировна | | | | |
|  |  |  |  |  |  |
| **Факультет** |  | Инфокоммуникационных Технологий | | | |
| **Группа** | К34212 | | |  | **Группа** |
| **Направление (специальность)** | | | | | 11.03.02. – Инфокоммуникационные технологии и системы связи |
| **Руководитель** | | Филянин Иван Викторович, аспирант ФПИ и КТ | | | |
| (Фамилия И.О., должность, ученое звание, степень) | | | | | |
| **Дисциплина** | | Сетевое программирование | | | |
| **Наименование темы** | | | | Анализ инструментов сетевой автоматизации | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№**  **п/п** | **Наименование этапа** | **Дата завершения** | | **Оценка и подпись**  **руководителя** |
| **Планируемая** | **Фактическая** |
| 1 | Анализ существующей научной литературы по теме | 7.12.2022 | 9.12.2022 |  |
| 2 | Создание плана курсовой работы | 10.12.2022 | 11.12.2022 |  |
| 3 | Составление основного текста курсовой работы | 20.12.2022 | 20.12.2022 |  |
| 4 | Создание презентации | 22.12.2022 | 23.12.2022 |  |
| 5 | Презентация курсовой работы перед аудиторией | 26.12.2022 | 26.12.2022 |  |

|  |  |  |
| --- | --- | --- |
| Руководитель | | Филянин Иван Викторович |
|  |  | (Подпись, дата) |
| Студент | Новожилова Анна Владимировна | |
|  |  | (Подпись, дата) |

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**АННОТАЦИЯ НА КУРСОВУЮ РАБОТУ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Студент** | Новожилова Анна Владимировна | | | | |
|  |  |  |  |  |  |
| **Факультет** |  | **Факультет** | | | |
| **Группа** | К34212 | | | **Группа** | К34212 |
| **Направление (специальность)** | | | | | 11.03.02. – Инфокоммуникационные технологии и системы связи |
| **Руководитель** | | Филянин Иван Викторович, аспирант ФПИ и КТ | | | |
| (Фамилия И.О., должность, ученое звание, степень) | | | | | |
| **Дисциплина** | | Сетевое программирование | | | |
| **Наименование темы** | | | | Анализ инструментов сетевой автоматизации | |

**ХАРАКТЕРИСТИКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)**

1. **Цель и задачи работы**

Определены руководителем Сформулированы при участии студента

Предложены студентом

Цель данной работы – провести анализ популярных систем автоматизации управления конфигурациями устройств в сети и сравнить их

1. **Характер работы**

Расчет

Моделирование

 Конструирование Другое

1. **Содержание работы**

В данном курсовом проекте представлены выявление основных сходств и различий систем автоматизации управления конфигурациями устройств в сети на основе имеющейся в открытых источниках теоретической информации и практического применения

1. **Выводы**

В ходе выполнения работы выполнен сравнительный анализ наиболее популярных инструментов для управления конфигурациями: Chef, Ansible и Terraform. При сравнении использовались данные, полученные при изучении теоретического материала и применении выбранных инструментов на практике.

|  |  |  |
| --- | --- | --- |
| Руководитель | | Филянин Иван Викторович |
|  |  | (Подпись, дата) |
| Студент | Новожилова Анна Владимировна | |
|  |  | (Подпись, дата) |

« » 2022г.

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 5](#_Toc122576921)

[ВВЕДЕНИЕ 6](#_Toc122576922)

[1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7](#_Toc122576923)

[1.1. СИСТЕМЫ УПРАВЛЕНИЯ КОНФИГУРАЦИЯМИ 7](#_Toc122576924)

[1.2. ANSIBLE 8](#_Toc122576925)

[1.3. CHEF 9](#_Toc122576926)

[1.4. TERRAFORM 12](#_Toc122576927)

[2.ПРАКТИЧЕСКАЯ ЧАСТЬ 15](#_Toc122576928)

[2.1. КОНФИГУРИРОВАНИЕ СЕТИ 15](#_Toc122576929)

[2.1.1. ANSIBLE 15](#_Toc122576930)

[2.1.2. CHEF 17](#_Toc122576931)

[2.1.3. TERRAFORM 18](#_Toc122576932)

[2.2. СРАВНИТЕЛЬНЫЙ АНАЛИЗ 21](#_Toc122576933)

[ВЫВОДЫ 23](#_Toc122576934)

[ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА 24](#_Toc122576935)

# ВВЕДЕНИЕ

Управление компьютерной сетью – трудоемкий и непрекращающийся процесс. Он включает в себя выполнение множества функций необходимых для контроля, планирования, выделения, внедрения, координации и [мониторинга](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BD%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%BE%D0%B9_%D1%81%D0%B5%D1%82%D0%B8) ресурсов компьютерной сети. При увеличении количества узлов в сети предприятия администрирование сети вручную системным администратором становится сложной задачей, возрастает шанс допустить ошибку в конфигурации, что с большой вероятностью приведет к финансовым потерям.

Для сокращения рисков на предприятиях все чаще применяют системы управления конфигурациями, позволяющие автоматизировать внесение изменений в работу отдельных узлов или целой сети. На сегодняшний момент в распоряжении сетевых администраторов представлены несколько наборов инструментов, позволяющих существенно облегчить развертывание, настройку и обновление конфигурации как небольших локальных сетей, так и достаточно масштабных объединений кластеров, насчитывающих десятки тысяч машин. В данной работе будут рассмотрены наиболее популярные на данный момент open-source решения для управления конфигурациями – Ansible и Chef, - а также новый инструмент, появившийся на рынке в 2014 году - Terraform. Сравнительный анализ и последующая классификация систем управления конфигурациями позволит уменьшить риск ошибки и временные затраты при выборе подходящего инструмента для администрирования.

Цель: Провести сравнительный анализ выбранных систем управления конфигурациями.

Задачи:

* Изучить теоретические основы систем управления конфигурациями
* Изучить особенности выбранных систем
* Изучить документацию выбранных систем
* Написать инструкции по установке сервиса на хост
* Сравнить работу и устройство выбранных систем
* Сделать выводы по результатам сравнения

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## СИСТЕМЫ УПРАВЛЕНИЯ КОНФИГУРАЦИЯМИ

Прежде чем детально рассматривать каждую из выбранных систем управления конфигурациями, необходимо изучить общее значение этого понятия. **Конфигурационное управление** (англ. software configuration management, SCM) в программной инженерии — комплекс методов, направленных на систематический учёт изменений, вносимых разработчиками в программный продукт в процессе его разработки и сопровождения, сохранение целостности системы после изменений, предотвращение нежелательных и непредсказуемых эффектов, формализацию процесса внесения изменений. Соответственно, **система управления конфигурацией** — программа или программный комплекс, позволяющий централизованно управлять конфигурацией множества разнообразных разрозненных операционных систем и прикладного программного обеспечения, работающего в них. Современные системы управления конфигурациями традиционно используется вместе с системами контроля версий и CI/CD инфраструктурой.

Большая часть систем управления конфигурациями может быть охарактеризована следующими функциями [3]:

1. Автоматизация. У каждой системы есть набор функций и синтаксис, определенный на одном из языков программирования, позволяющий реализовывать сценарии для автоматизации. Для большинство инструментов характерно использовать не полноценный язык программирования, а упрощенный синтаксис.
2. Идемпотентность. Системы управления конфигурациями имеют возможность контролировать уже созданные ранее на платформе ресурсы и выполненные задачи во избежание дублирования элементов архитектуры.
3. Хранение подробных данных о системе. Инструменты для управления конфигурациями предоставляют пользователю доступ к детальной информации об управляемой системе. Это позволяет писать скрипты, применимые к машинам с различными архитектурами.
4. Наличие системы шаблонов. Достаточно большое количество современных систем управления конфигурациями имеют собственную, интегрированную в них систему шаблонов. Шаблоны могут быть использованы для ускоренного написания скриптов. Шаблоны включают в себя неотъемлемые атрибуты языков программирования: переменные, циклы, условные операторы и т.д. Это помогает значительно упростить настройку компьютеров со схожей конфигурацией, но различными индивидуальными показателями.
5. Расширяемость. Практически любой инструмент для управления конфигурациями позволяет многократно использовать написанные скрипты в качестве модулей и библиотек, что дает пользователю возможность усложнять архитектуру серверов, не переписывая код.

В настоящее время популярность набирают системы, работающие по принципу Infrastructure as a Code. **Инфраструктура как код (IaC)** **-** это процесс управления и подготовки компьютерных центров обработки данных с помощью машиночитаемых файлов определений, а не конфигурации физического оборудования или интерактивных инструментов настройки [2]. Преимущества такого подхода заключаются в следующем [4]:

1. Использование ИТ-инфраструктуры в качестве средства быстрой доставки ценности
2. Сокращение усилий и рисков, связанных с внесением изменений в инфраструктуру
3. Предоставление пользователям инфраструктуры возможности получать необходимые им ресурсы, когда они в этом нуждаются
4. Предоставление общего инструментария для разработчиков, системных администраторов и других заинтересованных сторон
5. Создание систем, которые являются надежными, безопасными и экономически эффективными
6. Управление, безопасность и контроль соответствия инфраструктуры становятся прозрачными
7. Повышение скорости поиска неисправностей и устранения сбоев

Далее по тексту будут рассмотрены особенности работы каждой выбранной системы управления конфигурациями.

## ANSIBLE

Ansible - это программное обеспечение для автоматизации ИТ-процессов с открытым исходным кодом, управляемое из командной строки и написанное на языке программирования Python. Эта система управления конфигурациями может настраивать сервера в компьютерной сети, развертывать программное обеспечение и организовывать расширенные рабочие процессы для поддержки развертывания приложений, системных обновлений и т.д. [1].

Одной из отличительных особенностей Ansible является отсутствие необходимости устанавливать клиентские агенты на управляемых серверах, он использует протокол OpenSSH для транспортировки (с другими видами транспортировки и режимами извлечения в качестве альтернатив).

Платформа автоматизации Ansible структурирована в соответствии с концепцией узла управления и управляемого узла. Узел управления - это сервер, откуда выполняется Ansible. На него устанавливается программное обеспечение, и на нем находится командная строка, из которой пользователь дает указания. Управляемые узлы - это автоматизируемые устройства, например сервер Microsoft Windows.

Для автоматизации Linux и Windows Ansible работает путем подключения к управляемым узлам и отправки на них небольших программ, называемых "модулями Ansible". Эти программы написаны так, чтобы представлять собой ресурсные модели желаемого состояния системы. Затем Ansible выполняет эти модули и удаляет их по завершении. Эти модули спроектированы так, чтобы быть идемпотентными, когда это возможно, так что они вносят изменения в систему только при необходимости.

Для автоматизации сетевых устройств и других ИТ-устройств, где модули не могут быть выполнены, Ansible будет выполняться на узле управления. Поскольку Ansible не имеет агентов, он по-прежнему может взаимодействовать с устройствами, не требуя установки приложения или службы на управляемом узле. Чтобы увеличить производительность устройств, не имеющих возможности запускать модули, платформа автоматизации Ansible может распределять задания автоматизации по узлам выполнения, используя технологию, называемую automation mesh. В случае с облачными провайдерами Ansible также выполняет директивы на стороне управляющего узла, который, в свою очередь, взаимодействует с провайдером через API. Так как Ansible является инструментом с открытым исходным кодом, в него включена поддержка большого количества платформ и операционных систем.

Основными сильными сторонами Ansible являются простота и удобство использования, так как он позволяет писать скрипты автоматизации, используя читаемые человеком языки YAML или JSON. Он также уделяет большое внимание безопасности и надежности, отличаясь минимальным количеством изменяемых частей.

Недостатками Ansible являются недоработанный графический интерфейс, который позволяет осуществлять только часть функций системы, отсутствие отслеживания состояния машин, так как Ansible пользуется императивным подходом к управлению, а также отсутствие поддержки операционной системы Windows в качестве управляющего узла.

## CHEF

Chef — [система управления конфигурациями](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%84%D0%B8%D0%B3%D1%83%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5), написанная на [Ruby](https://ru.wikipedia.org/wiki/Ruby) ([клиентская часть](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))) и [Erlang](https://ru.wikipedia.org/wiki/Erlang) ([серверная часть](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5))), с использованием [предметно-ориентированного языка](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BC%D0%B5%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) для описания конфигураций. Используется для упрощения задач настройки и поддержки множества серверов и может интегрироваться в облачные платформы, такие как [Rackspace](https://ru.wikipedia.org/wiki/Rackspace) и [Amazon EC2](https://ru.wikipedia.org/wiki/Amazon_EC2), для автоматизации управления текущими процессами настройки новых серверов [8].

Конфигурация систем Chef включает в себя следующие элементы: основной сервер, на котором располагается хранилище особых файлов Chef – «поваренных книг» и «рецептов», узлы или ноды - удаленные сервера, к которым применяются изменения в конфигурации, и рабочая станция, которая отвечает за управление остальными узлами. Для работы с Chef требуются клиентские агенты, установленные на управляемых узлах. Программа-агент устанавливается на серверах с помощью утилиты knife, соединение между машинами осуществляется по SSН (англ. Secure Shell, сетевой протокол прикладного уровня, позволяющий производить удалённое управление операционной системой и туннелирование ТСР-соединений [9]). Дополнительной мерой безопасности является аутентификация управляемых узлов на рабочей станции, которая происходит с использованием сертификатов. Системный администратор, как пользователь программы Chef создаёт «рецепты», которые содержат описание настроек и конфигурации серверов и программ на них. Рецепты могут быть как написаны пользователем, так и скачаны на рабочую станцию из публичных репозиториев. Далее эти рецепты загружаются на сервер для хранения, к которому обращаются управляемые узлы, чтобы провести синхронизацию. По умолчанию синхронизация с рецептами происходит каждые тридцать минут, но эта настройка может быть изменена. «Рецепт» — это описание состояния системы, в котором она должна находиться в определенный момент времени, включая запущенные службы, установленные пакеты, созданные файлы [5]. Раз в установленное в настройках время Chef проводит проверку соответствия каждого из элементов системы сохраненной на сервере конфигурации, и если обнаружено несоответствие, приводит компьютер к требуемому состоянию.

В основном работа с Chef осуществляется по технологии клиент-серверного взаимодействия. В клиент-серверном режиме информация о текущем состоянии системы собирается с помощью клиентов, которые посылают на сервер всю возможную информацию о состоянии устройства. Серверная часть использует Apache Solr, чтобы проиндексировать свойства устройств. Также Solr предоставляет API для коммуникации с клиентом и дает ему возможность запрашивать информацию. «Рецепты» могут запрашивать эти свойства и использовать полученные данные для настройки хоста.

Помимо клиент-серверного режима в Chef доступен режим автономной конфигурации, который также называется «chef-solo».

Эта система управления конфигурациями обладает следующими преимуществами:

1. Быстрота развертывания. Если при написании скриптов соблюдаются определенные правила написания параметров конфигурации, добавление в сеть устройства или даже кластера серверов занимает незначительное количество времени по сравнению с настройкой сети вручную. Написание и выполнение скриптов на Chef у опытного сетевого администратора занимает не более часа.
2. Гибкость настроек. Так как Chef в своей работе использует хранилища «поваренных книг» Bookshelf, система управления конфигурациями имеет возможность создавать и одновременно хранить несколько сценариев поведения сети. Это позволяет изменять архитектуру и конфигурацию сети для выполнения задач разного рода за максимально короткое время. Такая возможность актуальна для тех сетей, которые требуют быстрой адаптации под нужды компании. Оперативное перераспределение ресурсной мощности сети является ключевым преимуществом и одной из причин популярности данного решения автоматизации.
3. Доступность. Решение для управления конфигурациями Chef предлагает различные варианты лицензии в зависимости от целей и частоты использования, что позволяет внедрять работу этой платформы в рабочий процесс значительного количества предприятий вне зависимости от их размера. Также доступна ограниченная по времени ознакомительная версия. широко распространено и доступно для широкого круга пользователей. Таким образом Chef остается инструментом, доступным широкому кругу пользователей.
4. Возможность использования на различных операционных системах. Рецепты Chef могут быть написаны и адаптированы для использования практически на любой операционной системе, что дает возможность поддерживать разнородные сети и позволяет использовать его на большом количестве клиентских машин.

Тем не менее, Chef также обладает рядом недостатков, в числе которых:

1. Чувствительность к ошибкам. Использование системы Chef в корпоративной сети подразумевает внимательность системного администратора и уверенные теоретические знания в конфигурировании и администрировании сетей. Chef не обладает мощным инструментом отслеживания логических ошибок, а сохраненные на сервере рецепты применяются автоматически, так что при наличии в коде ошибок значительно возрастает шанс столкнуться с полной потерей соединения, удалением данных или даже остановкой работы целого сегмента сети.
2. Безопасность. Важнейшей задачей при работе с Chef является защищенность рабочей станции. В случае пренебрежительного отношения к уязвимостям на контролирующем узле и несанкционированном проникновении в систему постороннего пользователя, злоумышленник получает полный контроль над всеми узлами в сети и может изменять их конфигурацию. Это может привести к серьёзному ущербу. При этом Chef обладает рядом серьёзных уязвимостей, связанных с использованными в его коде библиотеками, такими как LibYAML, Libcurl и другими [10].
3. Размер скриптов. Специфика написания рецептов в Chef делает конфигурационный код в среднем объемнее, чем в скриптах аналогичных систем управления конфигурациями, что является причиной возникновения дополнительных ошибок при написании и применении «книг рецептов», а также увеличивает время, необходимое для введения новой конфигурации в сеть.
4. Потребление большого количества ресурсов. На момент написания данной работы Chef уступает системам-конкурентам в производительности и потреблении системных ресурсов рабочей станции. Также Chef требует установки клиентского агента на каждый сервер в сети, что ведет к необходимости выделения памяти. Однако, оптимизация производительности является на данный момент приоритетным курсом компании при разработке новых версий Chef.

## TERRAFORM

HashiCorp Terraform - это инструмент, написанный на языке программирования Go и работающий по принципу Infrastructure as a Code, который позволяет пользователю определять как облачные, так и локальные ресурсы в удобочитаемых файлах конфигурации, которые можно изменять, повторно использовать и предоставлять общий доступ [6]. Непрекращающийся процесс синхронизации Terraform позволяет легко поставлять в сеть нужную инфраструктуру и управлять ею 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 и функции SaaS.

Terraform создает и управляет ресурсами на облачных платформах и других сервисах через собственные интерфейсы прикладного программирования (API). Широкий список облачных провайдеров и операционных систем, доступных в Terraform в виде модулей, также называемых провайдерами, позволяет работать практически с любой платформой или сервисом с помощью доступного API.

Пользователи могут взаимодействовать с поставщиками Terraform, объявляя ресурсы или вызывая источники данных. Вместо того, чтобы использовать императивные команды для предоставления ресурсов, Terraform использует декларативную конфигурацию для описания желаемого конечного состояния, тем самым делая повторное использование модулей проще и решая проблему случайного дублирования ресурсов. При выполнении скрипта Terraform на сервере в сети система будет выполнять от имени указанного пользователя CRUD-действия до тех пор, пока реальная конфигурация не будет соответствовать описанной в файле. Информацию о предыдущих изменениях конфигурации системы Terraform хранит в state-файле, что позволяет сократить время на выполнения каждого скрипта за счет игнорирования уже выполненных ранее изменений. [Инфраструктура как код](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D1%80%D0%B0%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BA%D0%B0%D0%BA_%D0%BA%D0%BE%D0%B4) может быть написана в виде модулей, что обеспечивает возможность повторного использования и поддержку кода.

Terraform, как система управления конфигурациями обладает следующими преимуществами:

1. Портативность. Terraform осуществляет поддержку всех популярных облачных провайдеров, что предоставляет пользователям широкий выбор платформ для разворачивания сети. Более того, эта система управления конфигурациями использует идентичные модули для всех поддерживаемых платформ, что позволяет легко переносить архитектуру из одного облачного провайдера в другой.
2. Открытый исходный код. Terraform является инструментом с открытым исходным кодом, что позволяет использовать его бесплатно даже в крупных корпоративных сетях. Более того, это обеспечивает постоянное обновление и улучшение существующих функций, так как система поддерживается не только разработчиками, но и активными пользователями.
3. Простота использования. Terraform использует для скриптов свой собственный язык программирования, основанный на YAML и отличающийся своей краткостью и легко воспринимаемым человеком форматом. Это позволяет легко разворачивать даже сложные, многоуровневые архитектуры.
4. Декларативный подход. Инструкции в файлах Terraform – это описание желаемого пользователем результата, а не строгая и обязательная последовательность действий. При выполнении файла Terraform выполняет только те инструкции, которые необходимы для достижения конечной цели, что экономит время и ресурсы системы.

К недостаткам системы можно отнести:

1. Новизна. По сравнению с остальными рассматриваемыми системами для управления конфигурациями Terraform является самым новым продуктом. Это неизбежно влечет за собой наличие все еще неисправленных ошибок и отсутствие некоторых функций, а также малое количество информации в сети.
2. Отсутствие условных операторов. Terraform на момент написания работы не поддерживает классический условный оператор. Для ветвления сценариев конфигурации используется тернарный оператор, что не всегда удобно и усложняет понимание написанного ранее кода.
3. Сложность совместного использования. Terraform в своей работе опирается на файлы состояний, которые по умолчанию хранятся локально в директории проекта. Таким образом, для управления конфигурацией с нескольких рабочих станций необходимо дополнительное программное обеспечение для общего доступа к файлу состояния. Более того, при запуске одного и того же модуля на разных локальных машинах (например, с разными версиями Terraform) имеется вероятность получить различный результат.
4. Специфичность сценариев использования. Существенный недостаток Terraform заключается в том, что программа разрабатывалась с расчетом на использования совместно с облачными провайдерами и не может быть использована для работы с кластерами выделенных серверов, в отличие от других систем управления конфигурациями. Эта особенность значительно сокращает количество сценариев использования утилиты.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## КОНФИГУРИРОВАНИЕ СЕТИ

В практической части данной работы будут более подробно рассмотрены алгоритмы установки и управления системами управления конфигурациями. Так как все инструменты обладают разным набором возможностей, для сравнения была выбрана простая задача, подходящая для выполнения с помощью всех рассматриваемых систем управления конфигурациями: установка сервиса nginx на заданных хостах. В качестве управляемого и управляющего узлов были использованы виртуальные машины с предустановленной на них операционной системой Ubuntu 20.04.

### 2.1.1. ANSIBLE

Ansible обладает широким набором возможностей и очень прост в установке и первичной настройке. Ansible не требует установки программного обеспечения на клиенты, а утилита для управляющего узла поставляется практически со всеми популярными дистрибутивами. Ansible был установлен командой apt-get install ansible.

Список узлов для управления в этой системе управления конфигурациями хранится в файле hosts. Узлы можно объединять в группы, а также обозначать для групп переменные.

Для данной задачи была добавлена новая группа coursework (см. Рисунок 1), в которую добавлен IP-адрес управляемого узла, а также указаны переменные – кредиты для подключения по ssh.

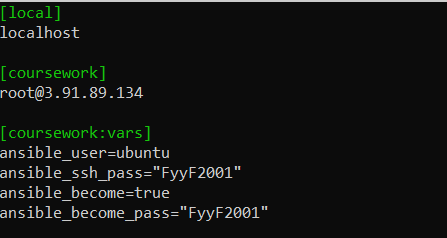


Рисунок 1 – Hosts файл

Управление узлами производится с помощью playbook, в котором указываются инструкции для выполнения. Разные инструкции могут выполняться для разных групп серверов. Был написан playbook, выполняющийся для узлов группы coursework (см. Рисунок 2). С помощью модуля apt устанавливается последняя версия nginx.

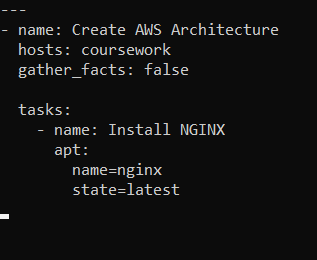


Рисунок 2 – Ansible playbook

После выполнения playbook командой ansible-playbook стал доступен отчет (см. Рисунок 3) об изменениях. Полнота отчета зависит от указанных при запуске команды параметров.

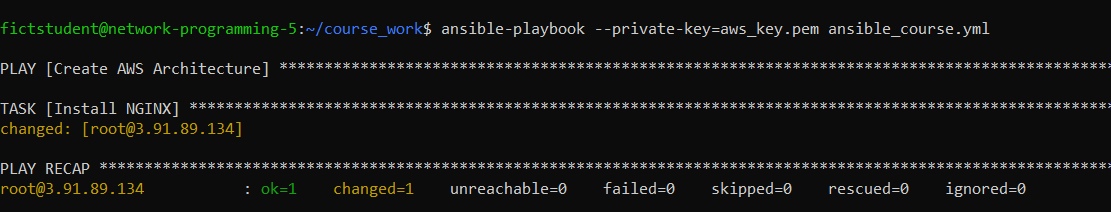


Рисунок 3 – Отчет о выполнении

В отчете статус выполненной инструкции указан как changed, что значит, что в управляемый узел были внесены соответствующие изменения. Увидеть их можно, введя публичный IP-адрес сервера в поисковую строку браузера. Приветственная страница nginx (см. Рисунок 4) указывает на то, что сервис был успешно установлен и запущен.



Рисунок 4 – Результаты выполнения

Важно упомянуть, что Ansible является императивной системой управления конфигурациями. Это значит, что при повторном запуске playbook она попытается повторно внести изменения, не учитывая текущую конфигурацию.

### 2.1.2. CHEF

Chef является самой сложной из рассмотренных систем управления конфигурациями. Полная версия Chef требует наличие как минимум трёх узлов для корректной работы: сервер, на котором хранятся все книги рецептов (сборники выполняемых инструкций), рабочая станция, откуда осуществляется управление, а также клиентский сервер. Так как в качестве задания для сравнения была выбрана очень простая инструкция, была настроена упрощенная версия chef-solo. Она обладает ограниченным функционалом, но позволяет совместить сервер и рабочую станцию.

Для работы с Chef требуются утилиты Ruby, Chef, knife-solo Berkshelf. Ruby есть в репозиториях всех популярных дистрибутивов, а все остальные утилиты необходимо загружать из сторонних источников. Вся работа с Chef ведется из кухни – директории, в которой создаются и хранятся все необходимые файлы. Кухня создается командами knife solo init и berks init. После создания кухни в ней появляется файл Berksfile, созданный для управления и загрузки cookbooks. Через этот файл добавляется cookbook-и ‘nginx’ и ‘nginx-wrapper’ (см. Рисунок 5), так как несмотря на отсутствие в документации упоминания об этом, стандартный рецепт и рецепт с установкой в cookbook-е ‘nginx’ больше не доступны.

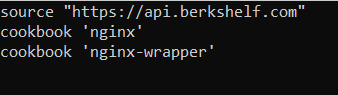


Рисунок 5 - Berksfile

Теперь, когда необходимые рецепты хранятся на сервере, их можно использовать в указаниях для узла. Все указания хранятся в папке nodes, а названия файлов соответствуют IP-адресу узла. Единственным обязательным пунктом в этом документе является run-list, который содержит в себе порядок рецептов для выполнения на узлах. Для выполнения установки достаточно выполнить рецепт по умолчанию из cookbook-а ‘nginx-wrapper’ (см. Рисунок 6).

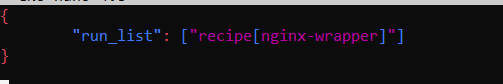


Рисунок 6 – Файл узла

Затем с помощью knife-solo была инициирована установка chef на клиента, а также первичное подключение клиента к серверу (см. Рисунок 7). Обнаружив настройки для своего IP-адреса, клиент выполнил их, приводя машину в указанное состояние.

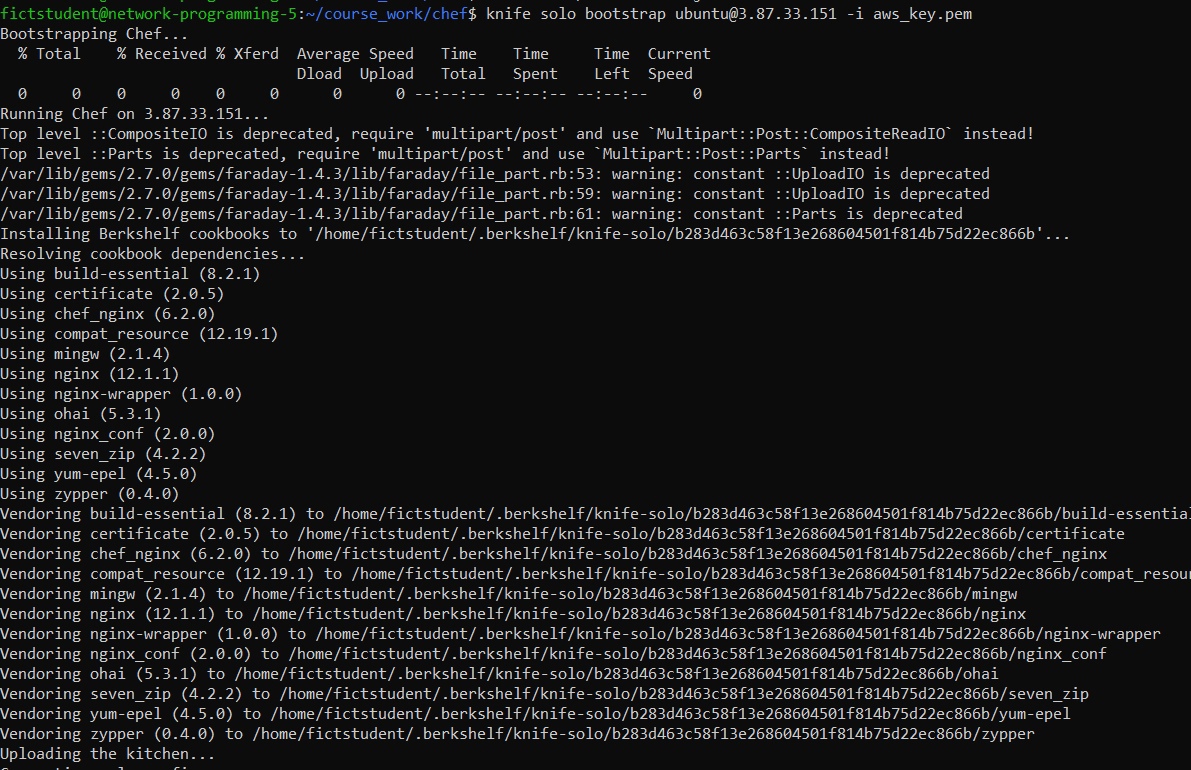


Рисунок 7 – Выполнение команды

После завершения выполнения команды по публичному IP-адресу хоста можно обнаружить приветственную страницу nginx (см. Рисунок 8), а значит установка прошла успешно.

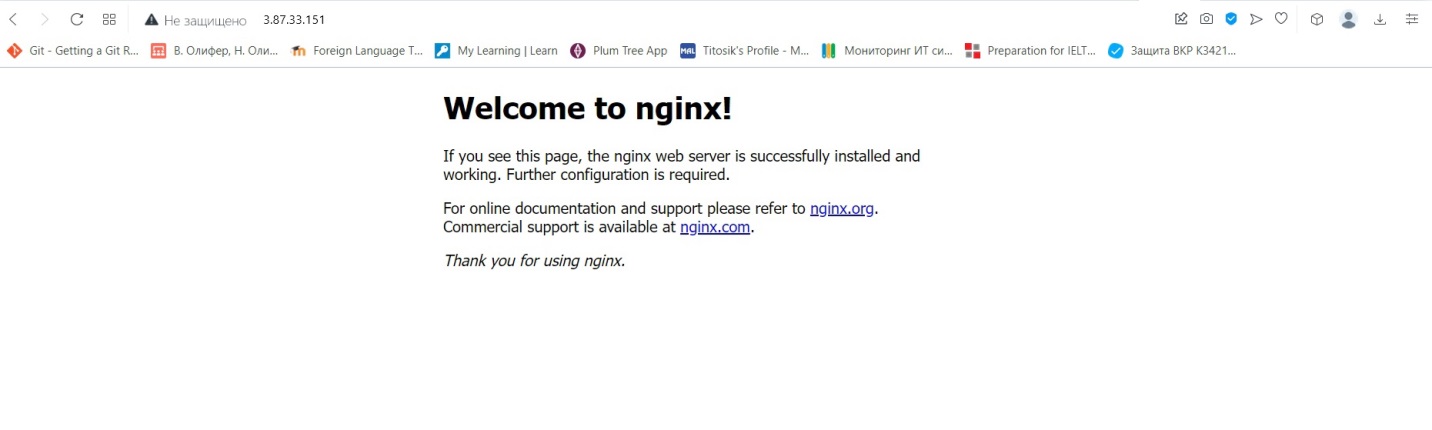


Рисунок 8 – Результат установки

### 2.1.3. TERRAFORM

Terraform на момент написания работы не доступен для использования на территории РФ, как и остальные продукты компании HashiCorp. По этой причине работа с данным инструментом производилась с помощью VPN, что незначительно, но усложняло его использование. Terraform доступен для установки во всех популярных дистрибутивах, так что установка утилиты была произведена с помощью команды apt-get install terraform.

Так как terraform в первую очередь создавался как система управления конфигурациями, совместимая с облачными провайдерами, установка nginx будет выполняться на экземпляр виртуальной машины в AWS. Terraform – декларативная система, которая при планировании изменений опирается на файл состояния, содержащий в себе все устройства, с которыми велась работа и их конфигурации. Так как в данной работе описывается первое взаимодействие с Terraform, во избежание дублирования экземпляров, в первую очередь необходимо передать в файл состояния информацию о существующем сервере с помощью команды terraform import.

Затем был создан файл с расширением tfvars (см. Рисунок 9). Этот файл содержит все переменные, используемые при осуществлении изменений в конфигурации. В этот файл были записаны данные, необходимые для подключения к инстансам: токены и SSH-ключ.

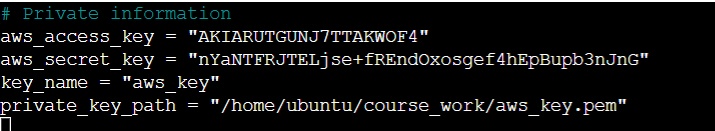


Рисунок 9 – Переменные подключения

Описание желаемой конфигурации хранится в файле module.tf (см. Рисунок 10), где был указан используемый провайдер, а также информация о виртуальных машинах.

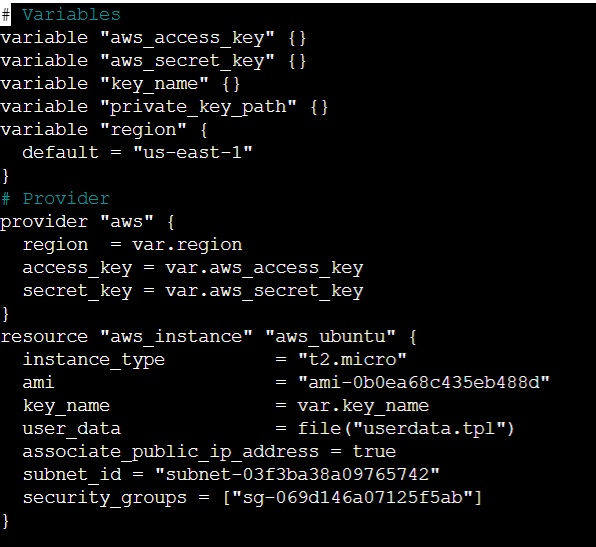


Рисунок 10 – Файл module.tf

Инструкции из файла, указанного в поле user\_data выполняются после создания или изменения экземпляра. В этом файле были указаны команды для установки nginx (см. Рисунок 11).

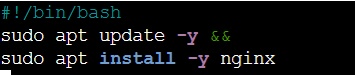


Рисунок 11 – Файл user\_data

После подготовки всех необходимых файлов была запущена команда terraform init (см. Рисунок 12), инициализирующая работу с terraform и позволяющая ему обнаружить все данные. Также была проверена правильность их написания с помощью команды terraform validate.

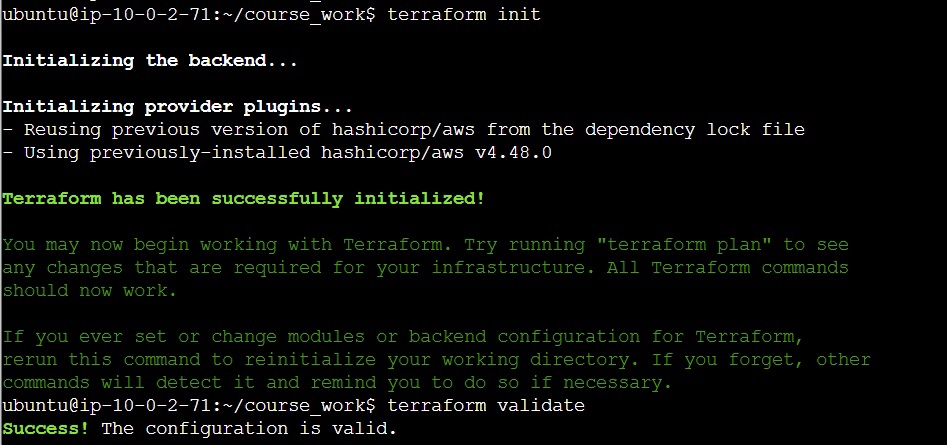
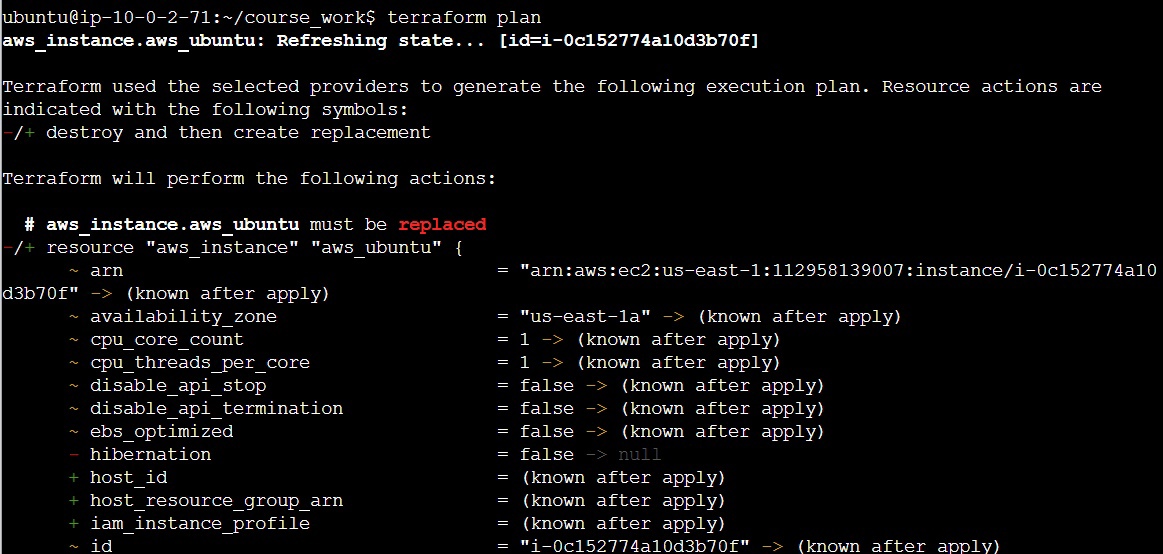


Рисунок 12 – Инициализация Terraform

Команда terraform plan (см. Рисунок 13) показывает, какие изменения будут применены к указанным экземплярам, чтобы привести их в желаемое состояние. При необходимости сервер может быть пересоздан. Это позволяет еще раз убедиться в корректности инструкций.

Рисунок 13 – Планирование изменений

Последней используемой командой является terraform apply (см. Рисунок 14), запускающая процесс применения изменений.

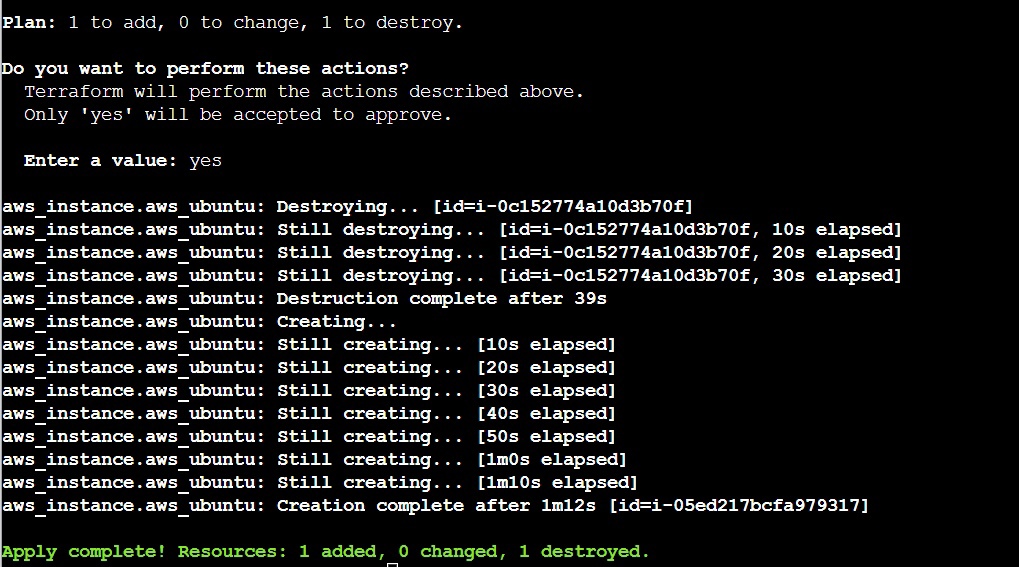


Рисунок 14 – Применение изменений

После завершения выполнения команды при открытии веб-страницы по публичному адресу сервера, можно увидеть приветственную страницу nginx (см. Рисунок 15). Следовательно, установка прошла успешно.

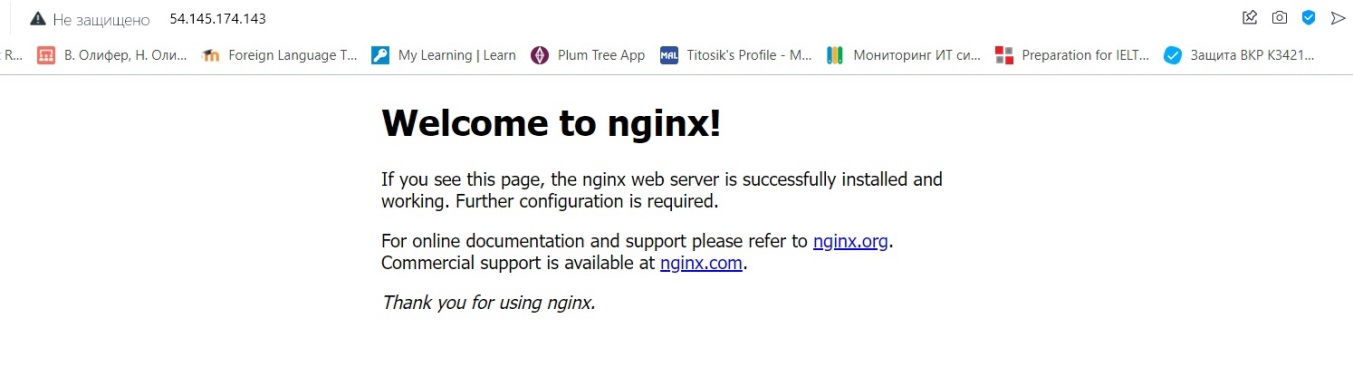


Рисунок 15 – Результат использования Terraform

## СРАВНИТЕЛЬНЫЙ АНАЛИЗ

На основе изученной теоретической информации, а также использования выбранных инструментов на практике был проведен сравнительный анализ систем управления конфигурации (см. Таблицу 1). Все системы рассматривались с точки зрения архитектуры, поддержки различных операционных систем, принципов работы, используемых языков программирования, актуальности и полноты документации. Ниже чуть подробнее рассмотрены результаты сравнения.

Все рассмотренные системы управления конфигурациями поддерживают установку на многие популярные операционные системы (по большей части Windows и Linux), что позволяет использовать их в разнородных сетях и не ограничивать выбор управляющего узла только одной операционной системой.

Ansible и Terraform не требуют установки дополнительного программного обеспечения, все управления ведется только с выделенного узла, оттуда инициируются изменения. Отсутствие специального программного обеспечения на клиентских серверах упрощает первичную настройку системы и позволяют писать сценарии для создания нового виртуального оборудования. Тем не менее, клиент-серверная архитектура Chef позволяет реализовать pull модель доставки изменений и сделать конфигурирование сети более автономным и своевременным.

Chef и Ansible используют для доставки изменений SSH, что, в отличии от Terraform, позволяет не ограничиваться доступными провайдерами, а осуществлять изменения на любом сетевом устройстве, поддерживающем протокол SSH.

Ansible и Chef являются императивными системами, то есть все указанные в инструкции шаги будут выполнены по порядку вне зависимости от текущего состояния узла. Декларативные системы при внесении изменений опираются не на четкие инструкции, а на желаемое состояние системы. Декларативная конфигурация упрощает настройку серверов, однако императивная предоставляет администратору полный контроль над осуществляемыми действиями.

Таблица 1. Сравнение систем

|  |  |  |  |
| --- | --- | --- | --- |
| Система | Ansible | Chef | Terraform |
| Архитектура | Безагентная | Клиент-серверная | Безагентная |
| Мультиплатформенность | Да | Да | Да |
| Протокол связи | SSH | SSH | API провайдера |
| Актуальная документация | Да | Нет | Да |
| Способ доставки конфигурации | Push | Push, pull | Push |
| Возможность создавать новые объекты | Да | Нет | Да |
| База данных | Нет | PostgreSQL | Нет |
| Язык разработки | Python | Ruby | Go |
| Язык инструкций | Yaml, Json | Json | HSL |

# ВЫВОДЫ

В ходе выполнения работы выполнен сравнительный анализ наиболее популярных инструментов для управления конфигурациями: Chef, Ansible и Terraform. При сравнении использовались данные, полученные при изучении теоретического материала и применении выбранных инструментов на практике.

Результаты сравнения показали, что, несмотря на схожий функционал, каждый из инструментов уместно применять в различных сценариях настройки инфраструктуры. Так, Terraform в большей степени предназначен для масштабирования сетевой инфраструктуры в облаке и плохо пригоден для установки сервисов и внутренней конфигурации устройств. Система управления конфигурациями Chef идеально подходит для поддержания сервисов в сети в актуальном состоянии и изменении внутренней конфигурации устройств, но не пригодна для развертывания новых серверов из-за клиент-серверной архитектуры. Ansible является универсальным инструментом, подходящим для использования практически в любом сценарии.

# ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

1. Ansible Community Documentation. Режим доступа: <https://docs.ansible.com/ansible_community.html> (дата обращения 13.12.2022)
2. Wittig, Andreas; Wittig, Michael (2016). Amazon Web Services in Action. Manning Press*.*
3. Селезнёва А.М. К вопросу о выборе системы управления конфигурациями для развертывания сервера организации, 2020.
4. Morris K. Infrastructure as a Code. O’Reily Media, 2016.
5. Chef Documentation. Режим доступа: <https://www.chef.io> (дата обращения: 13.12.2022)
6. Terraform Documentation. Режим доступа: <https://www.terraform.io> (дата обращения: 14.12.2022)
7. Puppet Documentation. Режим доступа: https://puppet.com/docs/ (дата обращения: 14.12.2022)
8. Stephen Nelson-Smith. Test-Driven Infrastructure with Chef.  O'Reilly Media, 2013.
9. RFC 4254 — The Secure Shell (SSH) Connection Protocol. Режим доступа: <https://www.rfc-editor.org/rfc/rfc4254> (дата обращения 14.12.2022)
10. Басыня Е.А. Самоорганизующаяся система управления трафиком вычислительной сети, 2014.