СОДЕРЖАНИЕ

[ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ](#_jda9v7qdxzr5) 3

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ](#_e6nzmdvdybma) 4

[ВВЕДЕНИЕ](#_84jv89enp99k) 5

[1 ОПИСАНИЕ СУЩЕСТВУЮЩИХ РЕШЕНИЙ И ИХ СРАВНЕНИЕ](#_fdxwglydxysa) 6

[2 РАЗВЕРТЫВАНИЕ ДЛЯ РЕШЕНИЯ ЗАДАЧИ…](#_faf2fv4bg2yl) 7

[ВЫВОД](#_pwy63o1yozvw) 8

[СПИСОК ЛИТЕРАТУРЫ](#_seqm156yhghn) 9

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Infrastructure as Code – Подход к управлению и автоматизации инфраструктуры, при котором инфраструктура описывается в виде программного кода, что позволяет создавать, изменять и удалять ресурсы автоматически и предсказуемо.

DevOps – Методология разработки программного обеспечения, которая объединяет процессы разработки (Development) и эксплуатации (Operations) для повышения эффективности и автоматизации процессов.

Software-Defined Networking – Архитектурный подход к сетям, при котором управление сетевыми устройствами осуществляется с помощью программного обеспечения, отделённого от аппаратной части.

Software as a Service – Модель предоставления программного обеспечения как услуги, при которой пользователи получают доступ к приложению через интернет без необходимости его локальной установки.

HashiCorp Configuration Language – Декларативный язык конфигурации, разработанный компанией HashiCorp, который используется для описания инфраструктуры и ресурсов в Terraform.

JavaScript Object Notation – Лёгкий текстовый формат для обмена данными, основанный на синтаксисе JavaScript, который используется для передачи данных между системами.

Create, Read, Update, Delete – Набор основных операций, выполняемых с данными в системе управления базами данных или других приложениях.

Continuous Integration – Процесс регулярной интеграции изменений кода от нескольких разработчиков в общую ветку репозитория с автоматическим тестированием и сборкой.

Secure Shell – Сетевой протокол, обеспечивающий защищённое подключение и управление удалёнными устройствами через зашифрованный канал связи.

Initialization File – Формат текстовых файлов конфигурации, состоящих из секций и пар "ключ-значение", используемый для хранения настроек.

YAML Ain’t Markup Language – Удобочитаемый формат сериализации данных, широко используемый для конфигурационных файлов.

Internet Protocol – Сетевой протокол, отвечающий за адресацию и передачу данных в сети Интернет.

Representational State Transfer Application Programming Interface – Стиль архитектуры программных интерфейсов, использующий HTTP-запросы для выполнения операций над ресурсами (например, получения данных, создания или удаления).

Access Control List – Список, определяющий права доступа к ресурсам, описывающий, кто и какие действия может выполнять с ними.

Terraform – Инструмент оркестрации инфраструктуры с открытым исходным кодом, позволяющий автоматически создавать, изменять и удалять ресурсы в облачных и мультиоблачных средах на основе декларативных конфигураций.

Ansible – Инструмент управления конфигурацией, автоматизации ИТ-процессов и развертывания приложений, который не использует агенты и работает через SSH или аналогичные методы подключения.

Playbook – В Ansible это файл, написанный на YAML, который описывает последовательность задач для выполнения на целевых узлах.

Terraform Provider – Подключаемый модуль в Terraform, который предоставляет интерфейсы для работы с конкретными ресурсами инфраструктуры (например, облаками, сетями или хранилищами данных).

Module – Независимый компонент в Ansible или Terraform, который выполняет определённую функцию, например управление ресурсами, настройку приложений или работу с внешними сервисами.

Declarative Approach – Подход, при котором описывается конечное состояние системы без указания пошаговой инструкции, как его достичь.

Imperative Approach – Подход, при котором указывается последовательность действий, необходимых для достижения заданного результата.

Mutable Infrastructure – Инфраструктура, где изменения применяются к существующим ресурсам.

Immutable Infrastructure – Инфраструктура, где для изменений создаются новые ресурсы, а старые удаляются, обеспечивая неизменность конфигурации.

Push Distribution – Модель, при которой управляющая система самостоятельно инициирует изменения на целевых устройствах.

Orchestration – Координация и управление взаимодействием между различными компонентами инфраструктуры, такими как сети, серверы и приложения.

Configuration Management – Процесс управления настройками и состоянием устройств и программного обеспечения для обеспечения их согласованности и работоспособности.

# 

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

IaC – Infrastructure as Code (Инфраструктура как код)

DevOps – Development and Operations (Подход к разработке и эксплуатации программного обеспечения)

SDN – Software-Defined Networking (Программно-определяемая сеть)

SaaS – Software as a Service (Программное обеспечение как услуга)

HCL – HashiCorp Configuration Language (Конфигурационный язык от HashiCorp)

JSON – JavaScript Object Notation (Универсальный формат обмена данными)

CRUD – Create, Read, Update, Delete (Операции создания, чтения, обновления и удаления данных)

CI – Continuous Integration (Непрерывная интеграция)

SSH – Secure Shell (Протокол для защищённого управления удалёнными устройствами)

INI – Initialization File (Формат текстовых файлов конфигурации)

IP – Internet Protocol (Интернет-протокол)

REST API – Representational State Transfer Application Programming Interface (Программный интерфейс на основе архитектурного стиля REST)

ACL – Access Control List (Список управления доступом)

ВВЕДЕНИЕ

Современные информационные технологии стремительно развиваются, и облачные вычисления стали основой для работы большинства организаций. Эффективное развертывание и управление облачной инфраструктурой является ключевым фактором, определяющим скорость и качество предоставления цифровых услуг. С ростом сложности архитектуры систем и увеличением числа используемых ресурсов ручные методы управления становятся не только неэффективными, но и рискованными.

Для решения этой проблемы применяются инструменты автоматизации, такие как Terraform, Ansible, Puppet и другие, которые предоставляют надежные подходы к оркестрации, развертыванию и управлению облачными ресурсами. Эти инструменты минимизируют риск человеческих ошибок, повышают масштабируемость систем и упрощают процессы настройки. Одной из основных концепций, лежащих в основе таких решений, является инфраструктура как код (Infrastructure as Code, IaC). IaC позволяет автоматизировать управление ИТ-инфраструктурой, ускоряя развертывание систем, снижая операционные затраты и обеспечивая стандартизацию и согласованность в управлении ресурсами.

IaC занимает центральное место в подходе DevOps, который объединяет команды разработки и эксплуатации для совместной работы над проектированием и конфигурацией инфраструктуры. Одной из ключевых задач в реализации IaC является выбор подходящего инструмента автоматизации. Несмотря на схожесть функционала, инструменты отличаются своими особенностями: одни упрощают конфигурационное управление, другие оптимизированы для оркестрации облачных сервисов. Например, Ansible специализируется на управлении конфигурацией, тогда как Terraform обеспечивает эффективную координацию облачных ресурсов.

В рамках данной работы проводится анализ и сравнение Ansible и Terraform, их ключевых характеристик, преимуществ и ограничений. Основное внимание уделяется выбору инструмента в зависимости от задач конкретного проекта, а также рассмотрению возможностей их совместного использования для реализации современных IT-инициатив.

# **1 ОПИСАНИЕ СУЩЕСТВУЮЩИХ РЕШЕНИЙ И ИХ СРАВНЕНИЕ**

## 1.1 Обзор Terraform

Terraform — это инструмент управления инфраструктурой с открытым исходным кодом, известный своей простотой в использовании. Его основная задача заключается в создании, масштабировании и управлении облачными ресурсами, а также в обеспечении согласованности состояния сети. В отличие от инструментов, ориентированных на настройку и управление программным обеспечением, Terraform не занимается установкой и конфигурированием ПО на существующих устройствах. Вместо этого его функциональность сосредоточена на создании, модификации и удалении серверов, а также других облачных ресурсов.

Terraform широко применяется в центрах обработки данных и программно-определяемых сетевых средах (Software-Defined Networking, SDN). Он одинаково эффективно работает как с низкоуровневыми компонентами инфраструктуры, такими как устройства хранения данных и сетевое оборудование, так и с высокоуровневыми сервисами, включая решения формата Software as a Service (SaaS).

С точки зрения управления состоянием, Terraform предоставляет возможность сопоставления текущего состояния ресурсов с описанной конфигурацией, сохраняя метаданные и улучшая производительность сети. Это делает его незаменимым инструментом для автоматизации управления современной облачной инфраструктурой.

### **Основные области применения Terraform**

Terraform находит применение как для управления ресурсами внешних поставщиков услуг, включая облачные платформы, так и для работы с собственными инфраструктурными решениями. Это делает его особенно полезным для многоуровневых приложений, таких как веб-серверы, взаимодействующие с базами данных. Благодаря способности Terraform моделировать зависимости между компонентами системы, он гарантирует, что уровень базы данных будет развернут и готов к использованию до запуска веб-серверов.

Terraform не привязан к конкретному облачному провайдеру, что позволяет ему эффективно работать в многоблочной среде, обеспечивая отказоустойчивость и гибкость. Использование единого конфигурационного файла предоставляет возможность управлять ресурсами сразу нескольких поставщиков, а также координировать зависимости между облаками.

Ещё одной важной областью применения Terraform является создание и управление временными инфраструктурными решениями, например, для демонстраций или тестовых сред. Его простота в настройке и использовании делает его идеальным выбором для быстрого развертывания облачных сетей. Кроме того, Terraform отлично справляется с управлением параллельными окружениями, что позволяет эффективно использовать его для тестирования, проверки исправлений и подготовки к выводу решений в эксплуатацию.

### **Принципы работы Terraform**

Terraform основывается на декларативном подходе, который позволяет описывать желаемое конечное состояние инфраструктуры без необходимости указания последовательности действий для его достижения. Инструмент работает на высоком уровне абстракции, предоставляя возможность описывать, какие облачные ресурсы и сервисы должны быть созданы и как они должны взаимодействовать. В отличие от низкоуровневого программирования, Terraform фокусируется на описании структуры и свойств инфраструктуры.

Конфигурации Terraform задаются с использованием языка конфигурации HashiCorp (HashiCorp Configuration Language, HCL) или формата JSON, при этом предпочтение отдается HCL благодаря его простоте и читаемости. Для работы с HCL не требуется специального опыта программирования, что делает его доступным для широкого круга пользователей.

С помощью HCL администраторы определяют поставщиков услуг и ресурсы сети. Каждый ресурс представляет собой конкретный элемент инфраструктуры, например виртуальную сеть или вычислительный узел. HCL предоставляет такие компоненты, как блоки, аргументы и выражения, которые упрощают процесс настройки:

* **Блоки** используются для логической группировки задач и обработки ошибок.
* **Аргументы** позволяют задавать значения идентификаторов — как статические, так и вычисляемые выражения.

При этом в HCL отсутствуют сложные структуры данных и элементы управления, что обеспечивает его интуитивную простоту.

Для выполнения реальных изменений Terraform использует провайдеры — программные модули, которые служат интерфейсом между инструментом и управляемыми ресурсами. Провайдеры, которые могут быть официальными или созданными сообществом, описывают типы ресурсов и источники данных, с которыми может работать Terraform. Пользователь обязан предварительно указать нужных провайдеров, чтобы Terraform смогла их установить. Большинство провайдеров связаны с определёнными инфраструктурными платформами, например облачными провайдерами, но некоторые из них могут быть универсальными утилитами.

Пользователи также могут разрабатывать собственные модули, которые позволяют переиспользовать конфигурации и обеспечивают гибкость. Файлы Terraform имеют расширение .tf и включают блоки для определения провайдеров и ресурсов.

### **Рабочий процесс Terraform**

Процесс работы с Terraform состоит из нескольких основных этапов, которые обеспечивают последовательное развертывание и управление инфраструктурой:

1. **Создание конфигурации**На первом этапе пользователи создают файлы конфигурации с использованием языка HCL (HashiCorp Configuration Language) в любом текстовом редакторе. Эти файлы описывают целевую инфраструктуру, включая необходимые компоненты и их конечное состояние.
2. **Планирование изменений**Выполнение команды terraform plan позволяет Terraform анализировать файлы конфигурации проекта и генерировать план действий. Этот план включает:
   * построение графа зависимостей между ресурсами,
   * возможность параллельного выполнения задач для независимых компонентов,
   * детальное отображение всех действий, которые Terraform намеревается выполнить.

Данный этап предоставляет техническим специалистам возможность предварительно ознакомиться с предполагаемыми изменениями, оценить их соответствие поставленным требованиям и внести необходимые корректировки в конфигурацию.

1. **Применение конфигурации**После утверждения плана команда terraform apply используется для применения конфигурации. Terraform разворачивает указанные ресурсы, синхронизируя фактическое состояние инфраструктуры с описанным в конфигурации.

Рабочий процесс Terraform предполагает постоянное взаимодействие между этапами создания и планирования конфигурации. Это позволяет пользователям проверять и уточнять файлы до их применения.

После выполнения конфигурации Terraform применяет операции создания, чтения, обновления и удаления (CRUD) для приведения ресурсов к заданному состоянию. Если в файлах конфигурации вносятся изменения, инструмент анализирует различия и создает детализированный поэтапный план, направленный на минимизацию сбоев в работе системы.

Дополнительные команды, поддерживающие рабочий процесс:

* terraform init — используется для предварительной подготовки необходимых провайдеров.
* terraform destroy — выполняет удаление созданных ресурсов и завершение работы сети.

Этот структурированный подход обеспечивает гибкость и контроль над процессом управления инфраструктурой, минимизируя риски и повышая производительность.

**Terraform и его взаимодействие с другими продуктами**

Несмотря на то, что Terraform не является инструментом управления конфигурацией, он может быть эффективно использован в сочетании с такими решениями для создания комплексных инфраструктурных систем. Terraform выполняет задачи на более высоком уровне абстракции, обеспечивая управление сетью и облачными ресурсами, в то время как инструменты управления конфигурацией, такие как Ansible или Puppet, могут использоваться для настройки отдельных устройств.

Кроме того, Terraform позволяет автоматизировать загрузку и развертывание программного обеспечения для управления конфигурацией, что делает его универсальным элементом в экосистеме DevOps.

Для упрощения процессов управления и улучшения совместной работы команд существует коммерческая версия инструмента — Terraform Cloud. Эта платформа предоставляет возможности централизованного рабочего пространства, что особенно актуально для команд, работающих над общей инфраструктурой. Terraform Cloud способствует повышению производительности, улучшая организацию процессов и интеграцию между участниками команды.

## 1.2 Обзор Ansible

Ansible, разработанный компанией Red Hat, представляет собой инструмент, основная цель которого заключается в автоматизации процессов информационных технологий. Он используется для автоматизации подготовки программного обеспечения, управления конфигурацией, развертывания приложений, а также создания и поддержания конвейеров непрерывной интеграции (Continuous Integration, CI).

Ansible обеспечивает интеграцию с облачными сетями и поддерживает взаимодействие с различными платформами. Инструмент совместим с большинством дистрибутивов Linux и может применяться как для управления устройствами под управлением Linux, так и для устройств, работающих на Windows.

При разработке Ansible основное внимание уделялось минимализму, согласованности, безопасности, надежности и простоте использования. Инструмент отличается лёгкостью установки и не требует от пользователей наличия специальных навыков программирования, что делает его доступным для широкого круга специалистов.

**Основные применения Ansible**

Ansible разработан для поддержки широкого спектра инфраструктурных платформ, включая простые серверы, виртуализированные среды, такие как гипервизоры, и облачные сети. Его гибкость обеспечивает плавную интеграцию с устаревшими системами и ранее существовавшими сценариями автоматизации, что делает его особенно подходящим для управления сложной и многогранной инфраструктурой, типичной для крупных предприятий.

Одной из ключевых особенностей Ansible является его поддержка идемпотентного поведения, гарантирующая, что узлы будут последовательно приводиться в одно и то же желаемое состояние независимо от количества применений конфигурации. Эта возможность имеет решающее значение для поддержания операционной согласованности и стандартизации поведения инфраструктуры в различных средах.

**Как работает Ansible**

Ansible работает без использования постоянных агентов на целевых узлах. Вместо этого он подключается к целевым системам с помощью SSH или других методов аутентификации и временно развертывает модули Python с помощью JSON. Эти модули, представляющие собой упрощенные программы, выполняемые на целевом узле, выполняют необходимые задачи и впоследствии удаляются по завершении выполнения. Такой подход сводит к минимуму использование ресурсов в целевой системе, когда она не находится под управлением. Как на управляющем, так и на целевом узлах должен быть установлен Python. Ansible не полагается на центральный сервер управления, поскольку любая машина, на которой установлен Ansible, может настраивать другие узлы. Средства управления доступом, такие как ключи авторизации, определяют, каким машинам разрешено управлять конкретными объектами.

Архитектура Ansible основана на использовании текста и не требует использования баз данных, демонов или внешних серверов. Такая простота повышает устойчивость, особенно при восстановлении после крупномасштабных сбоев.

Ansible поддерживает как декларативный, так и процедурный режимы работы. Он включает в себя собственный декларативный язык, позволяющий пользователям описывать желаемое конечное состояние системы или предоставлять пошаговые инструкции для достижения этого состояния. Ansible использует файлы инвентаризации, которые доступны для редактирования и управления версиями, для хранения информации об инфраструктуре. Эти файлы, записанные в формате INI или YAML, содержат список целевых узлов по имени хоста или IP-адресу. Одновременно можно использовать несколько файлов инвентаризации, а динамические запасы можно получать из внешних систем или местоположений.

Для эффективного управления узлы в файлах инвентаризации могут быть сгруппированы и вложены друг в друга. Использование диапазонов, переменных и псевдонимов еще больше упрощает организацию и обработку конфигураций инфраструктуры. Такая гибкость и модульность являются ключевыми для способности Ansible эффективно управлять разнообразными и сложными средами.

### Задачи, модули и учебные пособия Ansible

В Ansible действия по настройке в основном представлены в виде задач. Эти задачи соответствуют операциям, выполняемым в целевых системах. Задача может быть как отдельной специальной командой, так и вызовом определенного модуля. Модули в Ansible представляют собой отдельные файлы сценариев, обычно написанные на Python, хотя также могут использоваться такие языки, как Perl и Ruby. Каждый модуль обычно выполняет определенную функцию, например, управляет определенным приложением, и часто они объединяются в коллекции для упрощения доступа. Ansible включает в себя множество модулей по умолчанию.

Сборники в Ansible используются для группировки связанных задач и связанных переменных для эффективного выполнения. Эти сборники обычно написаны в удобочитаемом формате, таком как YAML, или с использованием шаблона Jinja, что упрощает их интерпретацию и модификацию. Playbooks может содержать требуемую конфигурацию сети, сведения о развертывании, учетные данные пользователя и информацию для входа в систему. Кроме того, playbooks может сопоставлять узлы из файлов инвентаря с ролями, которые представляют собой специализированные автономные playbooks, содержащие функции Ansible. Playbook выполняется последовательно, но он может включать циклы, управляющие операторы и обработчики событий для управления более сложными рабочими процессами.

Playbooks также предоставляет возможность запрашивать у администраторов вводимые значения, определять переменные и настройки по умолчанию и использовать результаты предыдущих команд для управления процессом выполнения задач настройки. Ansible playbooks поддерживает режим предварительной загрузки, позволяющий администраторам протестировать конфигурации перед фактическим развертыванием, гарантируя, что задачи будут выполняться должным образом. Это делает playbooks мощным инструментом для автоматизации и эффективного управления инфраструктурой.

**Ansible и другие продукты**

Ansible можно использовать различными способами, в зависимости от потребностей пользователя. Он может работать простым образом, используя специальные команды, что делает его подходящим для выполнения простых разовых задач. Однако он чаще используется с Ansible Playbooks, которые позволяют выполнять более широкий спектр инструкций, обеспечивая большую гибкость и контроль над задачами автоматизации.

Для корпоративных сред Ansible предлагает коммерческий продукт Ansible Tower. Это решение расширяет возможности Ansible, предоставляя полный набор функций, включая REST API, веб-консоль для управления, возможности планирования, списки контроля доступа (ACL) и функцию выполнения в один клик. Ansible Tower упрощает использование Ansible, превращая его в центральный узел автоматизации и оптимизации управления задачами в больших средах.

В дополнение к Ansible Tower доступны и другие коммерческие продукты, такие как Ansible Galaxy и Ansible Vault. Ansible Galaxy служит хранилищем для повторно используемых ролей, предоставляя пользователям предварительно настроенные компоненты для различных задач автоматизации, в то время как Ansible Vault позволяет шифровать конфиденциальные данные, обеспечивая безопасность файлов конфигурации и учетных данных в среде Ansible. Эти инструменты расширяют функциональность Ansible, делая его более надежным решением для управления сложными рабочими процессами автоматизации.

## 1.3 Сравнение решений

Terraform - это, прежде всего, инструмент для управления сервисами, оптимизированный для облачных и мультиоблачных сред. Его основная функция заключается в обеспечении того, чтобы инфраструктура достигала и поддерживала желаемое состояние. Terraform сохраняет это состояние и может восстановить заданную конфигурацию системы после перезагрузки. В отличие от инструментов управления конфигурацией, Terraform не фокусируется на настройке программного обеспечения на отдельных устройствах.

Ansible, с другой стороны, является инструментом управления конфигурацией, который специализируется на подготовке программного обеспечения и устройств, а также развертывании приложений в существующей инфраструктуре. Он работает на уровне отдельных устройств, независимо от сети, обеспечивая правильную работу каждого управляемого узла и его настройку в соответствии с назначением.

Несмотря на некоторое функциональное совпадение между Terraform и Ansible, у этих инструментов есть свои преимущества. Ansible включает в себя некоторые возможности для управления сервисами; например, его playbooks можно расширить для развертывания приложений в облачных средах. Кроме того, он предоставляет модули для интеграции с большинством крупных облачных провайдеров. Однако функции оркестровки Ansible менее надежны, чем у Terraform, особенно для управления взаимосвязанными и зависимыми приложениями или сервисами в сложных облачных сетях.

Чтобы провести различие между Terraform и Ansible, а также другими инструментами управления инфраструктурой как кодом (IaC), для оценки можно использовать несколько фундаментальных характеристик и возможностей. Такой подход позволяет глубже понять их соответствующие преимущества и сценарии, в которых каждый инструмент наиболее эффективен.

**Версии с открытым исходным кодом и коммерческие версии**

Как Terraform, так и Ansible являются бесплатными инструментами с открытым исходным кодом. Однако каждый из них также предлагает расширенные корпоративные версии или расширения, предназначенные для более крупных организаций, за определенную плату. Ansible предоставляет Ansible Tower, коммерческий продукт, который расширяет возможности автоматизации и добавляет функции для оптимизации операций. Аналогичным образом, Terraform предлагает Terraform Cloud, сервис, который упрощает и улучшает управление конфигурацией, с некоторыми основными функциями, доступными бесплатно, и премиум-функциями, доступными через платные аккаунты.

**Используемые технологии**

Terraform разработан на языке программирования Go и поддерживает файлы конфигурации, написанные на фирменном языке конфигурации HashiCorp (HCL) или JSON. Ansible, с другой стороны, построен с использованием Python и использует этот язык для настройки целевых узлов. Модули в Ansible могут быть написаны на Python, Perl или Ruby, а декларативные файлы конфигурации обычно создаются в формате YAML или INI. Примечательно, что и Terraform, и Ansible работают без использования агентов.

**Декларативный и императивный подходы**

Terraform использует исключительно декларативную методологию, позволяющую пользователям определять желаемое конечное состояние системы. Затем инструмент самостоятельно определяет необходимые шаги для достижения этой конфигурации. Ansible, поддерживая декларативную функциональность, также позволяет выполнять императивные процедуры за счет включения специальных инструкций и команд в свои скриптовые модули. Такая гибкость позволяет адаптироваться к различным вариантам использования, но создает дополнительную сложность, поскольку Ansible по своей сути не поддерживает запись конечного состояния системы.

**Изменяемые и неизменяемые конфигурации**

Возможность изменения конфигурации является еще одним отличительным фактором между этими двумя инструментами. Изменяемые конфигурации могут быть изменены напрямую, в то время как неизменяемые конфигурации требуют перезагрузки или полной перестройки целевой системы для применения изменений. И Terraform, и Ansible поддерживают изменяемые конфигурации, однако Ansible превосходит их в этой области благодаря своей универсальности, позволяющей легко изменять и исправлять конфигурации. Terraform, оптимизированный для облачных сред, лучше подходит для неизменяемых подходов. Во многих случаях удаление и воссоздание облачного ресурса с новой конфигурацией более эффективно, чем попытки внесения изменений.

### Распространение по принципу "один против одного"

Как Terraform, так и Ansible используют модель распространения по принципу "один против одного", которая позволяет активно отправлять конфигурации на целевые устройства, не требуя от устройств автономной загрузки конфигураций.

### Внешние ресурсы и поддержка сообщества

Преимущества Terraform заключаются в более развитой библиотеке готовых модулей, которая предоставляет пользователям надежный репозиторий для быстрого внедрения. Хотя Ansible предоставляет доступ к своему репозиторию Galaxy, этот ресурс обычно требует большего ручного вмешательства. Оба инструмента поддерживаются обширными и активными сообществами пользователей, что облегчает обмен знаниями и сотрудничество.

### Графические интерфейсы пользователя (GUI)

Ни один из инструментов не предлагает комплексных решений с графическим интерфейсом. Ansible включает базовый графический интерфейс в свой корпоративный продукт Ansible Tower, хотя его функциональность несколько ограничена. В отличие от этого, в Terraform полностью отсутствует собственный графический интерфейс, вместо него используется интерфейс командной строки и вспомогательные инструменты.

### Итоги

Этот сравнительный анализ подчеркивает уникальные преимущества и области применения Terraform и Ansible, помогая принимать обоснованные решения по их интеграции в рабочие процессы автоматизации инфраструктуры.

## 

# **2** ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ

## 2.1 Развертывание сервиса в облаке с помощью Terraform

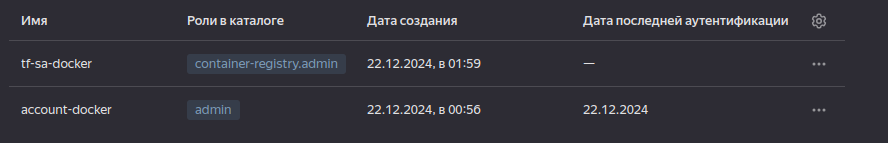
Для выполнения поставленной задачи было использовано следующее ПО:

* OS: Arch Linux
* Editor: VS Code
* Docker
* IaC: Terraform

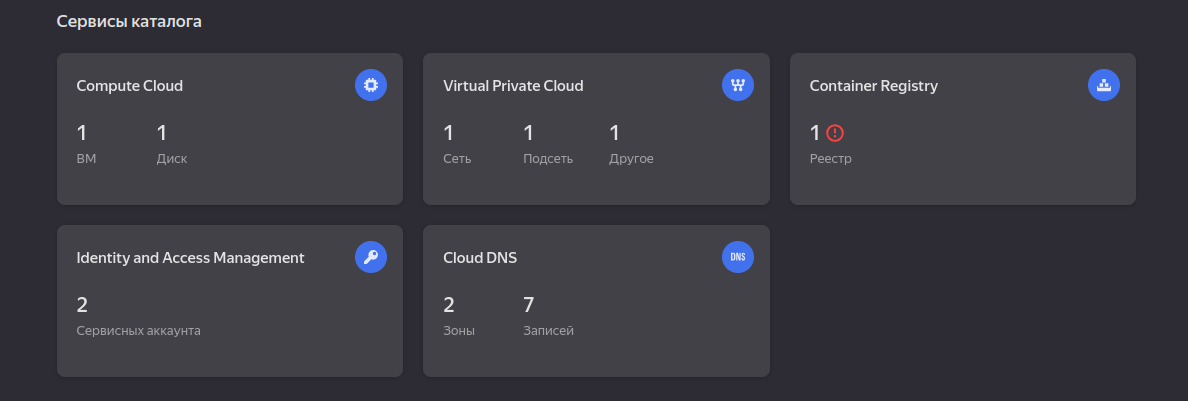
Чтобы показать практическое применение описанных в работе технологий, был создан докер-контейнер с простым веб-приложением. Для развертывания этого приложения при помощи Terraform был выбран отечественный облачный провайдер Yandex Cloud.

В первоначальную настройку облака входило:

1. Создание аккаунта
2. Создание каталога docker
3. Создание пользователя с привилегиями admin
4. Создание пользователя с привилегиями для управления container registry
5. Генерация IaM ключей для пользователя

После того, как облако было настроено, необходимо было собрать инфраструктуру для непосредственного развертывания веб-приложения. Для выполнения этой задачи были предприняты следующие шаги:

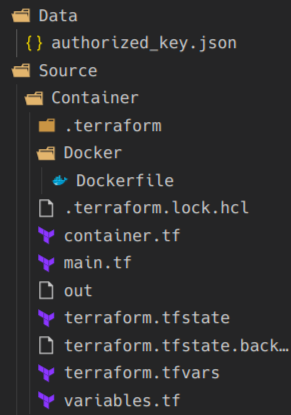
1. Определение провайдера Yandex Cloud (См. приложение А1)
2. Назначение привилегий для пользователя, управляющего container-registry (См. приложение А2)
3. Создание container-registry (См. приложение А3)
4. Создание и настройка сети (См. приложение А4)
5. Создание образа виртуальной машины с Ubuntu для управления контейнерами (См. приложение А5)

Следующим шагом после создание инфраструктуры стало загрузка в облако самого docker-образа веб-приложения (См. приложение В1). Для этого были предприняты следующие действия:

1. Авторизация через docker credential manager с Yandex Cloud
2. Присвоение тэга docker-образу для загрузки в Yandex Cloud
3. Загрузка docker-образа в Yandex Cloud

Для внедрения best practices проект был разбит на несколько файлов:

* **main.tf**, хранящий основные настройки terraform и провайдеров
* **container.tf**, хранящий конфигурацию создаваемой инфраструктуры
* **variables.tf**, в котором декларируются используемые переменные
* **terraform.tfvars**, в котором присваиваются необходимые значения переменным

Но при этом стоит отметить что файл **.tfstate** хранился локально, что в production среде влечет за собой риски безопасности.

Таким образом Terraform наглядно продемонстрировал свою пригодность и удобство для решения задач CI/CD.

## 

## 2.2 Использование Ansible для управления инфраструктурой VMware

## В рамках проекта по автоматизации управления виртуальной инфраструктурой была проведена работа по внедрению Ansible для взаимодействия с платформой VMware. В качестве хоста для запуска Ansible использовался сервер с Ubuntu 22.04. Ansible продемонстрировал свою пригодность для управления виртуальными машинами и хостами ESXi благодаря наличию модулей, основанных на библиотеке pyVmomi (Python SDK для работы с VMware vSphere API).

#### Установка и настройка среды

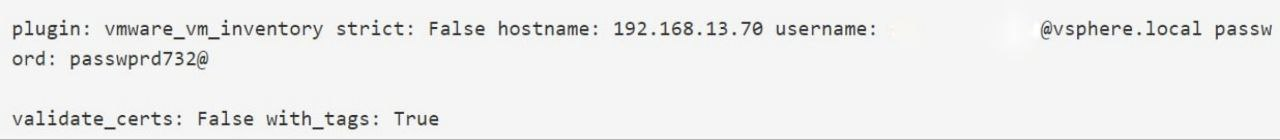
#### Для начала работы была выполнена установка библиотеки pyVmomi с использованием менеджера пакетов pip.

#### 

#### После этого в конфигурационном файле ansible.cfg был активирован плагин vmware\_vm\_inventory, позволяющий использовать динамическую инвентаризацию ресурсов VMware.



## Для подключения к vCenter и хостам ESXi был создан YAML-файл с необходимыми переменными (vmware\_vars.yml), включающими параметры аутентификации, адреса серверов и настройки сертификатов.

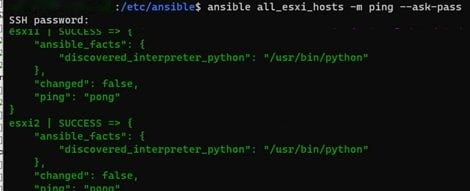


## Получение инвентаризации и управление ESXi-хостами

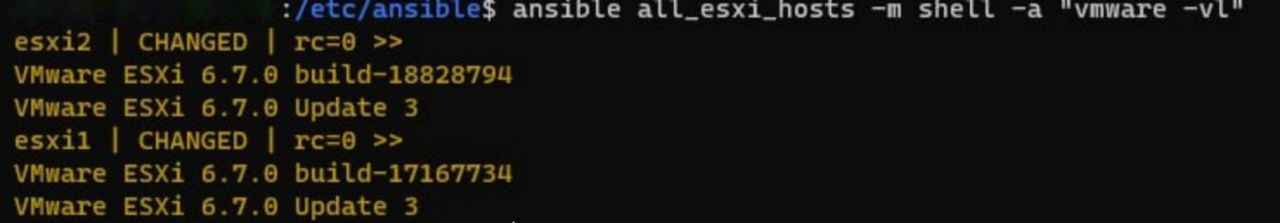
## Далее была выполнена настройка инвентаризации для получения списка виртуальных машин. Это позволило централизованно управлять всеми ресурсами через единую точку. Для работы с хостами ESXi был дополнительно создан файл /etc/ansible/hosts, в котором указаны IP-адреса и параметры доступа.



## На следующем этапе была проверена доступность всех ESXi-хостов с помощью модулей Ansible. Это подтвердило корректность настроек инвентаризации и доступность SSH-подключений.



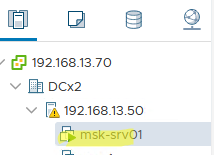
## Для тестирования управления хостами ESXi выполнены команды в режиме ad-hoc, включая запрос версии ESXi и запуск отдельных shell-команд.

Создание и тестирование Playbook для VMware

В рамках работы был разработан Playbook vm\_start.yml, предназначенный для запуска виртуальных машин на платформе VMware. В Playbook используются параметры подключения, хранящиеся в vmware\_vars.yml, а также модуль vmware\_guest\_powerstate, позволяющий изменять состояние виртуальных машин. Приложение А6.

После запуска Playbook была проверена работа через интерфейс vSphere Client, что подтвердило успешное выполнение задачи.





Автоматизация управления службами ESXi

Отдельное внимание уделено разработке Playbook для управления службами на ESXi-хостах. Для этого была установлена коллекция модулей community.vmware. В Playbook использовался модуль vmware\_host\_service\_manager, позволяющий включать и отключать службы, такие как ntpd. Приложение А7.

В результате выполнения данного Playbook на выбранных хостах ESXi служба ntpd была успешно активирована, что подтвердило работоспособность автоматизации.

## В ходе работы было выявлено ограничение: в бесплатной версии VMware Hypervisor (ESXi) доступ к API предоставляется только на чтение, что накладывает ограничения на автоматизацию задач.

# **ВЫВОД**

И Terraform, и Ansible обладают неоспоримыми преимуществами, что делает их подходящими для различных вариантов использования в рамках автоматизации инфраструктуры.

Сильные стороны Terraform

Terraform известен своим удобным интерфейсом и надежными возможностями планирования. Его интеграция с Docker является существенным преимуществом, поскольку Docker эффективно управляет задачами настройки, устраняя недостаток внимания Terraform в этой области. Однако Terraform может создавать значительные накладные расходы и демонстрировать непрозрачное поведение. Процессы, которые он использует для приведения целевых систем в желаемое состояние, не всегда понятны, и даже окончательная конфигурация может быть непрозрачной.

Сильные стороны Ansible

Ansible отличается превосходными функциями безопасности и управления списками контроля доступа (ACL), предлагая более совершенное решение, которое легко интегрируется в традиционные системы автоматизации. Ansible интуитивно понятен и прост в эксплуатации. Тем не менее, он менее эффективен при организации сложных сервисов, управлении взаимозависимыми приложениями и обработке логических зависимостей между компонентами инфраструктуры.

Дублирование и совместное использование Terraform и Ansible

Несмотря на различия, Terraform и Ansible в значительной степени совпадают по функциям. Во многих случаях можно эффективно использовать оба инструмента. Важно отметить, что они не являются взаимоисключающими; оба могут использоваться одновременно в рамках одной сети. Оптимальный выбор часто зависит от конкретных требований инфраструктуры и желаемых результатов.

Рекомендации по выбору инструмента

Выбор между Terraform и Ansible должен основываться на всестороннем понимании текущей сетевой среды и предполагаемых целей. Для контейнерных решений, предназначенных для предоставления программного обеспечения в облачных средах, Terraform, вероятно, будет более подходящим вариантом. И наоборот, Ansible обеспечивает повышенную гибкость и контроль для управления смешанными устаревшими сетями с различными приложениями. Это особенно выгодно для сценариев, в которых необходимо поддерживать детальный контроль над отдельными устройствами, используя альтернативные методы управления базовыми службами.

Учитывая продолжающуюся эволюцию обоих инструментов, их возможности могут продолжать сближаться или расширяться, потенциально предлагая более комплексные решения. Следовательно, наиболее подходящий выбор для данной сети сегодня может измениться в будущем. Рекомендуется периодически пересматривать свои потребности в инфраструктуре и возможности этих инструментов, чтобы обеспечить соответствие долгосрочным целям.

# **СПИСОК ЛИТЕРАТУРЫ**

1. <https://developer.hashicorp.com/terraform/docs>
2. https://registry.terraform.io
3. <https://yandex.cloud/ru/docs>

ПРИЛОЖЕНИЕ

## Приложение А1

provider "yandex" {

service\_account\_key\_file = "../../Data/authorized\_key.json"

cloud\_id = var.cloud\_id

folder\_id = var.folder\_id

zone = var.default\_zone

}

## Приложение А2

resource "yandex\_iam\_service\_account" "sa" {

folder\_id = var.folder\_id

name = "tf-sa-docker"

}

resource "yandex\_resourcemanager\_folder\_iam\_member" "registry-sa-role-images-puller" {

folder\_id = var.folder\_id

role = "container-registry.admin"

member = "serviceAccount:${yandex\_iam\_service\_account.sa.id}"

}

## Приложение А3

resource "yandex\_container\_registry" "docker-registry" {

name = "docker-registry"

folder\_id = var.folder\_id

}

## Приложение А4

resource "yandex\_vpc\_network" "docker-vm-network" {

name = local.network\_name

}

resource "yandex\_vpc\_subnet" "docker-vm-network-subnet-a" {

name = local.subnet\_name

zone = var.default\_zone

v4\_cidr\_blocks = ["192.168.1.0/24"]

network\_id = yandex\_vpc\_network.docker-vm-network.id

}

## Приложение А5

resource "yandex\_compute\_disk" "boot-disk" {

name = "bootvmdisk"

type = "network-hdd"

zone = var.default\_zone

size = "10"

image\_id = local.vm\_image\_id

}

resource "yandex\_compute\_instance" "docker-vm" {

name = local.vm\_name

platform\_id = "standard-v3"

zone = var.default\_zone

service\_account\_id = "${yandex\_iam\_service\_account.sa.id}"

resources {

cores = 2

memory = 2

}

boot\_disk {

disk\_id = yandex\_compute\_disk.boot-disk.id

}

network\_interface {

subnet\_id = "${yandex\_vpc\_subnet.docker-vm-network-subnet-a.id}"

nat = true

}

metadata = {

user-data = "#cloud-config\nusers:\n - name: ${local.username}\n groups: sudo\n shell: /bin/bash\n sudo: 'ALL=(ALL) NOPASSWD:ALL'\n ssh\_authorized\_keys:\n - ${file("${local.ssh\_key\_path}")}"

}

}

## Приложение А6



## Приложение А6

