Reza Faezi

Transitioning from Terraform to OpenTofu: A Comparative Study and Migration Guide

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

25 July 2024

Abstract

Author:	Reza Faezi
Title:	Transitioning from Terraform to OpenTofu: A Comparative Study and Migration Guide
Number of Pages:	35 pages
Date:	25 July 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Smart IoT & Cloud Computing
Supervisors:	Kimmo Sauren, Head of Major

Infrastructure as Code (IaC) has transformed the management and configuration of computing infrastructure in information technology (IT), promoting efficiency, scalability, and reliability through code automation. This study aims to examine the implications of transitioning from Terraform, a widely used Infrastructure as Code (IaC) tool, to OpenTofu, an emerging open source alternative. The study explores the similarities and differences between the two tools, the challenges organization face during migration, and the best practices for ensuring a smooth transition.

The study was conducted through online research, analysing the core features, architecture, and workflows of both tools. It includes a detailed comparative analysis, focusing on areas such as feature sets, ease of use, licencing, cost implications, and security considerations. It also covers the migration process in-depth, offering a guide to converting Terraform scripts to OpenTofu, along with tools and strategies for testing and validating the new configurations.

The results of the study show that while Terraform and OpenTofu share foundational elements, the primary difference lies in their licencing: OpenTofu is entirely open source, unlike Terraform's new Business Source Licence (BSL). The study highlights migration challenges, such as dependency management and compatibility issues, and outlines strategies to address them.

In conclusion, the study benefits organizations seeking to switch from a proprietary to an open source IaC tool by providing practical insights and migration guidance. It recommends a careful, incremental migration approach and emphasizes the importance of comprehensive testing and stakeholder engagement. Further research is suggested to explore performance comparisons and develop frameworks for migration.

Keywords: IaC, Terraform, OpenTofu, BSL licence, HashiCorp

The originality of this thesis has been checked using Turnitin Originality Check service.

Contents

List of Abbreviations

AMI:	Amazon Machine Image
AWS:	Amazon Web Services
BSL:	Business Source Licence
CI/CD:	Continuous Integration and Continuous Delivery
CLI:	Command-Line Interface
DevOps:	Development and Operations
EC2:	Elastic Compute Cloud
HCL:	HashiCorp Configuration Language
IaC:	Infrastructure as Code
IT:	Information Technology
JSON:	JavaScript Object Notation
RBAC:	Role Based Access Control
SaaS:	Software as a Service
S3:	Amazon Simple Storage Service
tf:	Terraform manifest file

1 Introduction

In the landscape of IT, IaC has become a key practice for managing and setting up computing infrastructure. By using code to automate the setup and configuration process, IaC makes IT operations more efficient, scalable, and reliable. This approach allows infrastructure to be treated in the same way as application code, which enables consistency and reduces the potential human error. IaC is a crucial part of today’s development and operations (DevOps) practices, creating a need for tools that can effectively set up and manage infrastructure configurations.

One of the tools that has become popular in the IaC space is Terraform, which is widely used for its flexibility and extensive ecosystem. However, recent changes in Terraform’s licencing, specifically the transition to a Business Source Licence (BSL) by HashiCorp (Dadgar, 2023), have raised concerns within the community about the long-term openness and accessibility of the tool.

In response to these concerns, OpenTofu has emerged as a promising open source alternative to Terraform. OpenTofu aims to uphold the principles of openness and community-driven development, making sure that the advantages of IaC remain accessible to everyone. As a fork of Terraform, OpenTofu stays compatible with existing Terraform workflows but also brings in its own features and improvements to better serve its users' needs. (OpenTofu, n.d., Manifesto.)

The transition from Terraform to OpenTofu brings both benefits and hurdles for organizations. On the one hand, OpenTofu allows continued use of IaC without the limitations of a BSL. On the other hand, moving to a new tool involves various technical, operational and organizational factors. It is important for organizations to understand the key differences and similarities between Terraform and Opentofu, be aware of potential risks of migrations and follow the best practices to ensure a smooth transition.

This study was designed to help organizations considering switching from Terraform to OpenTofu. It will explore how the two tools are alike and how they differ, talk about the challenges that might come up during the switch, and suggest best practices to help ensure a successful transition.

To guide this research, the following questions are addressed:

1. What are the key differences and similarities between Terraform and OpenTofu in terms of features and usability?

2. What challenges are associated with migrating from Terraform to OpenTofu, and how can they be overcome?

3. What best practices should be followed to ensure a successful migration process?

This study focuses specifically on comparing Terraform and OpenTofu and the process of migrating between them. It will not cover other IaC tools or broader DevOps practices beyond these two. The study is based on the current features and community support of Terraform and OpenTofu, so any future changes or updates to these tools will not be included. The goal is to offer practical guidance based on the current state of these tools.

2 Background and Literature Review

2.1 Historical Development of Terraform

Terraform has become one of the leading tools for organizations looking to manage their IaC. It simplifies the process of handling infrastructure across different cloud platforms by allowing users to write their configurations in a straightforward, readable format using HashiCorp Configuration Language (HCL). Terraform then takes care of building and managing the infrastructure based on these configurations.

Organizations using cloud services like AWS, Azure, and Google Cloud have moved toward Terraform for its ability to manage even the most complicated setups. Its strong community support, along with features like modular design and state management, has made it a top pick for automating infrastructure.

However, in 2023, HashiCorp made a major change by switching Terraform’s licence to a BSL in an effort to protect its intellectual property. This change raised concerns about the long-term openness of the tool, leading many to call for a truly open source alternative (Sanders, 2023).

2.2 Emergence of OpenTofu

In response to HashiCorp's shift in licencing, OpenTofu was created as a community-driven alternative to Terraform, aiming to preserve the open source values at its core. OpenTofu was designed to be a free, open source IaC tool, which provides users with full access and the ability to actively contribute to its development. While it remains compatible with Terraform, allowing users to continue working with their existing setups, OpenTofu also introduces new features and improvements. However, it is important to note that OpenTofu has stated on their website that, over time, the two projects are expected to diverge

in terms of features (OpenTofu, n.d., What Are the Differences between OpenTofu and Terraform?).

The launch of OpenTofu is a key moment in the IaC space, highlighting the importance of open source values in managing infrastructure. By branching off from Terraform, OpenTofu's developers want to create a neutral, community-centered platform where people can work together to improve the tool without being tied to a proprietary licence. OpenTofu’s focus on openness and transparency has attracted support from organizations and individuals who appreciate the advantages of open source software.

2.3 Review of Existing Literature on IaC Tools

The concept of IaC has been widely explored in both academic and industry settings. IaC fundamentally changes the way infrastructure is managed by treating it as software, which brings significant improvements in automation, consistency, repeatability, and scalability (Michalowski, 2024).

Research has examined various aspects of IaC, including its benefits, the challenges involved in implementation, and best practices. Studies have also highlighted the role of IaC in supporting DevOps practices, reducing human error, and accelerating the deployment and management process. Various IaC tools, such as Terraform, AWS CloudFormation, Ansible, and Puppet, have been reviewed in the literature, with a focus on their features, usability, and platform support. However, there has been less comparative analysis, particularly when it comes to newer tools like OpenTofu versus more established ones like Terraform.

2.4 Gaps in the Existing Literature This Study Aims to Fill

The existing literature on IaC tools lays a solid groundwork for understanding their benefits and challenges in today's IT environments. However, there are a few key areas that this study will explore:

1. Research on OpenTofu: While there is ample information on IaC tools such as Terraform, research on OpenTofu is sparse. This study will explore OpenTofu’s features, and usability, comparing it to Terraform to highlight its strengths and differences.
2. Migration Challenges: There is a lack of detailed guidance on transitioning from Terraform to OpenTofu. This study will explore the technical hurdles of migration and offer best practices to help organizations make the switch smoothly.
3. Best Practices for Successful Migration: Identifying best practices to ensure a successful migration process is part of addressing the challenges associated with the migration, ensuring a smooth transition.

By tackling these gaps, this study seeks to improve the understanding of the changing IaC landscape and provide practical advice for using OpenTofu in managing infrastructure.

3 Terraform Overview

Terraform is a broadly used IaC tool developed by HashiCorp that allows users to define, provision, and manage infrastructure using a declarative configuration language. This makes it easy to deploy infrastructure consistently and reliably across different cloud platforms and services.

This chapter dives into Terraform's core concepts, architecture, strengths, limitations, common use cases, its ecosystem and community support.

3.1 Detailed Explanation of Terraform

As seen in Figure 1, the typical Terraform workflow involves a practitioner writing infrastructure as code. Terraform then processes this code through two key steps: Plan and Apply. The Plan phase allows the user to preview the changes that will be made while the Apply phase implements these changes across various cloud providers and services, such as AWS, Google Cloud, Microsoft Azure, and Kubernetes. The result is the automated deployment and management of infrastructure across these platforms.

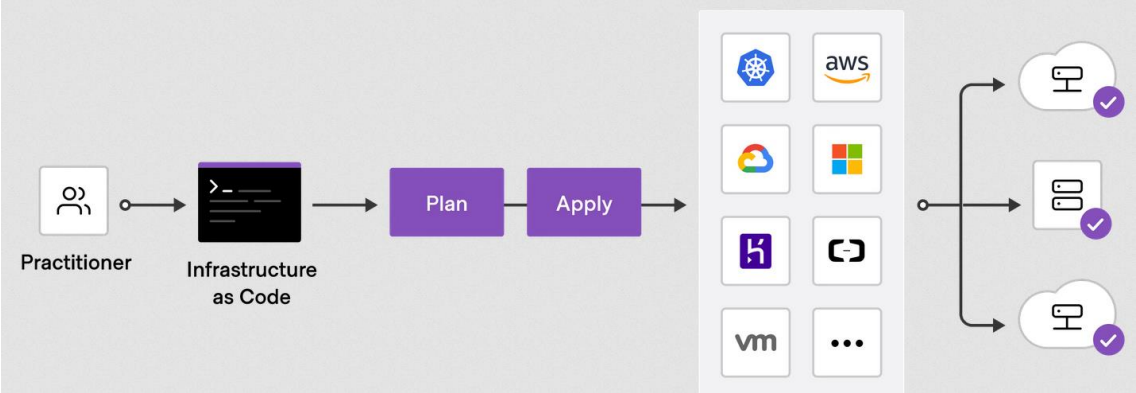

Figure 1: Terraform overview (Terraform, n.d., What is Infrastructure as Code with Terraform?)

3.1.1 Core Concepts

Terraform operates by using a set of concepts that is essential to how it works:

- **Providers**: Providers are the interface between Terraform and the target platform or services (e.g., AWS, Azure, Alibaba Cloud, Google Cloud). The provider configuration file as can be seen in Appendix 1, is shown, with the AWS provider configured to operate in the us-west-2 region. Terraform works with a wide range of providers, including public cloud services, Software as a Service (SaaS) platforms, and private infrastructure, making it a flexible tool for both multi-cloud and hybrid environments (Terraform, n.d., Providers).
- **Resources**: Resources are the building blocks that represent infrastructure components such as virtual machines, databases, and networks. Users define these resources in configuration files, as shown in Appendix 2, where both the Amazon Machine Image (AMI) and instance type have been configured. Terraform interacts with the relevant provider to create, update, or remove these resources as necessary (Terraform, n.d., Resource Blocks).
- **State**: Terraform keeps a state file that tracks the current state of the infrastructure it manages. This file is essential for monitoring changes and ensuring Terraform understands what resources exist in the real environment. By using the state file, Terraform can apply updates incrementally, modifying only what is needed to keep the infrastructure in sync. (Terraform, n.d., Manage resources in Terraform state).
- **Modules**: Modules are reusable configurations that help organize and share Terraform code. By grouping multiple resources into a single unit, modules simplify the management of complex infrastructure as can be seen in Figure 2. They can be shared across teams or sourced from a public registry, making it easier to scale and maintain one’s Terraform setups (Terraform, n.d., Modules overview state).

Figure 2: Terraform overview (Faezi, 2024)

3.1.2 Architecture

Terraform uses a declarative approach, where users specify the desired state of their infrastructure rather than detailing the steps to achieve it. The typical Terraform workflow involves several key steps, as illustrated in Figure 3, which shows how Terraform interacts with cloud service providers and manages infrastructure through configuration files and providers.

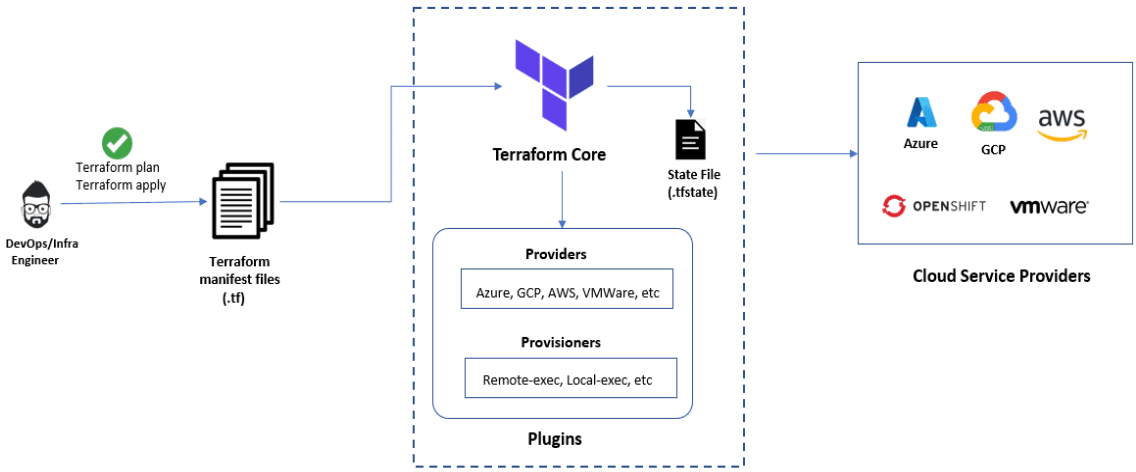

Figure 3: Architecture of Terraform (Rapaka, 2023)

The process starts with a DevOps or Infrastructure engineer writing Terraform manifest files (.tf). These manifest files are then used in combination with Terraform's commands to provision infrastructure:

1. **Write**: Users write configuration files using HashiCorp Configuration Language (HCL) or JavaScript Object Notation (JSON) to define the desired state of their infrastructure, including resources and their attributes.
2. **Initialize**: The terraform init command initializes the working directory, downloading necessary providers and modules. This step ensures that Terraform has access to the appropriate plugins for interacting with cloud providers.
3. **Plan**: The terraform plan command generates an execution plan, which shows the changes that will be made to the infrastructure based on the current state and the configuration files. It is a dry run that allows users to review and confirm the changes before applying them.
4. **Apply**: The terraform apply command executes the planned changes and updates the infrastructure. Terraform interacts with the providers to provision or modify resources and updates the state file to reflect the new infrastructure state.
5. **Destroy**: When infrastructure is no longer needed, the terraform destroy command can be used to tear down all managed resources, cleaning up the environment.

3.1.3 Strengths and Limitations

Terraform is a widely used IaC tool that offers numerous advantages for managing and automating infrastructure.

Strengths:
Below are some of Terraform´s key strengths.

- **Multi-Provider Support**: Terraform’s ability to manage resources across multiple providers simultaneously is one of its strongest features. This makes it ideal for multi-cloud strategies or hybrid environments as users can manage different infrastructure types using a single tool.

- **Declarative Syntax**: Terraform’s declarative language simplifies infrastructure management by focusing on the desired end state rather than the specific steps needed to achieve it.
- **Modularity and Reusability**: Using modules allows teams to create reusable infrastructure components while improving the efficiency and standardization. Modules can be shared or pulled from public registries, which promotes collaboration.
- **State Management**: Terraform’s state file tracks changes incrementally which reduces downtime and minimizes accidental modifications.

Limitations:
Despite Terraform´s strengths, it also has certain limitations that users should consider when adopting it. The following are some of the main limitations of Terraform.

- **State File Management:** The state file is crucial for Terraform but handling it can sometimes be tricky. If the file becomes corrupted or is not shared properly between teams, it can cause issues with managing infrastructure.
- **Immutability of Resources**: Terraform often recreates resources instead of updating them directly, which helps maintain consistency. This can lead to downtime when resources need to be replaced during updates.
- **Learning Curve**: For beginners, getting familiar with Terraform’s syntax and managing a state file can be difficult. While Terraform is powerful, its declarative approach can make troubleshooting harder, especially in more complex environments.

3.2 Common Use Cases

Terraform is widely used across various industries and for multiple infrastructure management scenarios. Some common use cases include the following:

- **Multi-Cloud Infrastructure Management**: Many organizations turn to Terraform to manage resources across various cloud providers, such as AWS, Azure, and Google Cloud. Its capability to operate in multi-cloud environments makes it an appealing option for companies looking for flexibility and wanting to avoid vendor lock-in.
- **Automated Provisioning**: Terraform automates the provisioning of infrastructure resources by minimizing the need for manual setup and by ensuring consistent, repeatable deployments. This is especially beneficial for environments that need to scale rapidly, like web applications and microservices architectures.
- **CI/CD Integration**: Terraform is commonly integrated into continuous integration and continuous delivery (CI/CD) pipelines. This allows infrastructure changes to be automatically applied during the software delivery process which ensures that the infrastructure evolves in sync with the application it supports.
- **Disaster Recovery**: Terraform's IaC approach enables organizations to quickly rebuild their infrastructure after a disaster. By using predefined configurations, resources can be redeployed in minutes, helping maintain business continuity.
- **Compliance and Auditing**: By defining infrastructure as code, Terraform helps enforce compliance and auditing standards. Organizations can apply security policies and ensure infrastructure configurations follow best practices by reviewing and versioning Terraform code.

4 OpenTofu Overview

OpenTofu is a rising alternative in the IaC space, developed as a community-driven response to Terraform's licencing changes. It offers an open source, transparent solution that remains compatible with existing Terraform setups. This chapter will dive into OpenTofu's key features, architecture, advantages, and challenges, and compare it with Terraform.

4.1 Core Concepts

OpenTofu shares many of its core concepts with Terraform, as it was directly forked from the original tool. These are described in subsequent sections.

4.1.1 Providers

Like Terraform, OpenTofu uses providers to connect with different platforms and services as shown in Appendix 3. These providers translate OpenTofu's configuration files into API calls for services like AWS, Azure, Google Cloud, and on-premise environments. They act as a bridge between OpenTofu and the infrastructure, thus enabling users to manage various resources through a single interface (OpenTofu, n.d., a).

4.1.2 Resources

Like Terraform, in OpenTofu resources represent the infrastructure components being managed, such as virtual machines, databases, storage, and networks. These resources are defined in configuration files, and OpenTofu uses providers to create, update, or remove them based on the desired state set by the user (OpenTofu, n.d., b). For example, the basic resource configuration for an AWS instance using OpenTofu as illustrated in Appendix 4, where the virtual machine is specified with an AMI ID and an instance type.

4.1.3 State Management

Like Terraform, OpenTofu manages infrastructure by keeping a record of the current state of resources. This is done through a state file, which reflects the actual setup of the infrastructure. When changes are made in OpenTofu's configuration, they are applied to ensure the real infrastructure matches the desired setup. The state file is crucial for tracking changes and making sure updates are applied incrementally and accurately (OpenTofu, n.d., c).

4.1.4 Modules

Like Terraform, OpenTofu uses modules to group multiple resources into reusable components. This helps simplify complex infrastructure setups and encourages collaboration by allowing teams to share standardized configurations. Modules can be sourced from public repositories or created in-house to meet specific organizational needs. (OpenTofu, n.d., d.)

4.2 Architecture and Workflow

OpenTofu shares a similar architecture and workflow with Terraform, highlighting their common roots. The process starts with creating configuration files that outline the desired infrastructure, followed by initialization, planning, and finally applying those configurations to the target infrastructure.

Workflow Steps:

1. **Write**: Users define their desired infrastructure using the same declarative syntax (HCL or JSON) as in Terraform. These configuration files specify the resources, providers, and variables necessary for building and managing infrastructure.
2. **Initialize**: The tofu init command sets up the working directory by downloading the required provider plugins and preparing the

environment for further actions. This ensures OpenTofu has the necessary tools to interact with cloud providers and other services.

3. **Plan**: The tofu plan command generates a proposed execution plan, illustrating how the current infrastructure will change based on the configuration. This step is crucial for understanding the impacts of any modifications before they are applied.
4. **Apply**: The tofu apply command executes the plan by provisioning or updating resources according to the configuration files. OpenTofu then updates the state file to reflect the new infrastructure state.
5. **Destroy**: If the infrastructure needs to be removed, the tofu destroy command can be used to delete all resources managed by OpenTofu.

(OpenTofu, n.d., Command Line Interface: CLI Commands.)

4.3 Strengths and Limitations

OpenTofu is a strong alternative in the IaC landscape, especially for organizations prioritizing open source principles and community driven development.

4.3.1 Strengths:

Below are some of OpenTofu´s key strengths.

- **Open Source Commitment**: One of OpenTofu's greatest strengths is its unwavering commitment to being genuinely open source. This means users can freely use, modify, and distribute the tool without the limitations of proprietary licences.
- **Compatibility with Terraform**: As a fork of Terraform, OpenTofu retains compatibility with existing Terraform configurations. This allows organizations to transition their infrastructure without needing to completely rewrite their IaC scripts, making the migration smoother.
- **Community-Driven Development**: OpenTofu is developed by a community of contributors, ensuring that it remains attuned to user needs

and can evolve organically, without being influenced by commercial interests.

4.3.2 Limitations

Despite its strengths, OpenTofu has certain limitations that stem from its relatively new status and reliance on community support. Below are some of the main limitations of OpenTofu.

- **Smaller Ecosystem**: As a newer tool, OpenTofu has a smaller ecosystem compared to Terraform. Although it can leverage Terraform's existing providers and modules, its own registry and community resources are still developing, which might slow its adoption among larger enterprises.
- **Limited Enterprise Support**: Unlike Terraform, which provides enterprise-grade support and features through HashiCorp, OpenTofu relies solely on community contributions and support. This may be a drawback for organizations that need guaranteed support levels.
- **Potential Lag in Feature Parity**: Because OpenTofu is a fork of Terraform, any significant advancements or new features introduced in Terraform may take time to appear in OpenTofu, depending on the community's efforts and priorities.

4.4 Differences and Similarities with Terraform

Understanding the similarities and differences between Terraform and OpenTofu provides valuable insights into their usability, adaptability, and appeal for different users and organizations. Below are the main similarities and differences.

4.4.1 Similarities

Terraform and OpenTofu share key features which are listed below.

- **Declarative Syntax**: OpenTofu and Terraform share the same configuration language (HCL), which makes it easy for existing Terraform users to switch to OpenTofu without having to learn a new syntax.
- **State Management**: Like Terraform, OpenTofu keeps track of the infrastructure state in a state file. This enables incremental updates and allows for rollbacks when needed.
- **Providers and Resources**: Both OpenTofu and Terraform utilize providers to manage infrastructure resources. OpenTofu supports the same providers and can handle the same types of resources, ensuring users can continue working with the platforms they already know.
- **Modularity**: Both tools emphasize the importance of modules for organizing and reusing infrastructure configurations, which helps enhance scalability and maintainability in complex environments.

4.4.2 Differences

Below are the main differences between Terraform and OpenTofu.

- **Licencing**: The most significant difference between OpenTofu and Terraform is in their licencing. OpenTofu is completely open source, allowing unrestricted use and distribution. In contrast, Terraform is now governed by the BSL, which imposes some limitations on usage and distribution. This makes OpenTofu a more appealing choice for organizations that value open source software.
- **Community-Driven Development**: While Terraform's development is overseen by HashiCorp, giving it control over the tool’s direction and features, OpenTofu thrives on community-driven development. This means that contributors from around the world can participate in shaping the project, with decisions made collectively, ensuring a more inclusive and responsive development process.

4.5 Ecosystem and Community Support

OpenTofu is still in its early stages but is quickly gaining traction with increasing community involvement and ecosystem support. As an open source project, it actively encourages contributions from developers, organizations, and enthusiasts, fostering the development of a dynamic and vibrant ecosystem around the tool.

4.6 Module and Provider Ecosystem

Although OpenTofu directly builds on Terraform’s providers and modules, its ecosystem is slowly expanding as more users and contributors get involved. This includes a growing collection of community-contributed modules and integrations with different platforms, which enhance its functionality and versatility. (Github, 2024).

4.7 Community Engagement

The OpenTofu project benefits from a diverse community of developers and users committed to advancing open source infrastructure management. Online forums, GitHub repositories, and social media platforms serve as vibrant spaces for collaboration and support, enabling users to exchange ideas, report issues, and contribute code. As the project evolves, this community will likely play a crucial role in shaping its direction and maintaining its significance in the IaC landscape.

5 Comparative Analysis

This chapter provides a detailed comparative analysis between Terraform and OpenTofu. The analysis covers key aspects such as features, cost implications, ease of use, security considerations, and integration capabilities. This comparison is essential to understanding the practical differences between these two IaC tools and how organizations can choose between them based on their needs.

5.1 Feature Comparison Between Terraform and OpenTofu

5.1.1 Providers and Modules

Both Terraform and OpenTofu provide strong support for a wide range of providers and modules, enabling users to manage resources across platforms like AWS, Azure, Google Cloud, and on-premises infrastructure.

- **Terraform**: Terraform offers one of the largest provider ecosystems, with hundreds of official and community-maintained providers. Its module registry is extensive, featuring numerous reusable components contributed by both HashiCorp and the community.
- **OpenTofu**: As a direct fork of Terraform, OpenTofu inherits the same provider and module ecosystem, making it fully compatible with Terraform’s providers and modules. However, as OpenTofu evolves, it may develop its own unique community and registry over time.

5.1.2 Licencing

Licencing is where the two tools significantly differ.

- **Terraform**: In 2023, Terraform transitioned to the BSL, limiting certain commercial uses, such as distributing Terraform as a service, while allowing personal and internal organizational use. (HashiCorp, 2023).
- **OpenTofu**: OpenTofu, remains fully open source under the Apache 2.0 licence. This gives users complete freedom to use, modify, and redistribute it without commercial restrictions, making it an appealing choice for organizations that prioritize open source software and licencing flexibility. (OpenTofu, n.d., Manifesto).

5.1.3 Workflow and Command Structure

Since OpenTofu is based on Terraform, its workflow and commands are almost identical:

- **Terraform**: Key commands like terraform init, terraform plan, terraform apply, and terraform destroy are used to manage infrastructure in a clear and predictable way, thanks to Terraform's declarative workflow.
- **OpenTofu**: OpenTofu maintains the same workflow and commands (e.g., tofu init, tofu plan), so users can transition from Terraform to OpenTofu seamlessly without learning a new process.

5.1.4 State Management

Both tools use state management to track infrastructure changes.

- **Terraform**: Terraform allows state files to be stored locally or remotely, enabling version control and detecting infrastructure drift. HashiCorp's enterprise version offers additional features beyond state locking, such as enhanced collaboration tools (e.g., private module registries, role-based access control, cost estimation, policy as code) and team governance features.
- **OpenTofu**: OpenTofu mirrors Terraform's state management, supporting both local and remote storage. As an open source tool, OpenTofu

provides these features without requiring users to pay for additional enterprise services, though it may lack some advanced features offered by Terraform's paid version. (Terraform, n.d., Support for Terraform Enterprise).

5.2 Cost Implications

Cost is a significant consideration, particularly for large-scale infrastructure management.

- **Terraform**: While Terraform's core tool is free, HashiCorp offers enterprise features like advanced state management, role-based access control (RBAC), and auditing which come at a cost and are valuable for large organizations.
- **OpenTofu**: OpenTofu is completely free and open source under the Apache 2.0 licence, making it a cost-effective alternative. While it lacks a paid enterprise version, organizations can use external or open source tools to replicate some of Terraform's advanced features, making it a good option for budget-conscious teams.

5.3 Ease of Use and Learning Curve

Ease of use is essential for teams adopting a new tool.

- **Terraform**: Terraform's ecosystem is mature, with extensive documentation, tutorials, and a wide range of community resources. However, for beginners, learning IaC concepts and managing complex modules can be challenging.
- **OpenTofu**: Since OpenTofu is almost identical to Terraform in terms of syntax and workflow, the learning curve remains the same. Users familiar with Terraform will find it easy to switch, but as a newer tool, OpenTofu's documentation and resources are still growing. However, its compatibility with Terraform helps mitigate this issue.

5.4 Security Considerations

Security is vital, especially when managing infrastructure in sensitive environments.

- **Terraform**: Terraform includes built-in security features such as state encryption and secrets management, with enterprise-level role-based access controls (RBAC) available through HashiCorp's paid offerings.
- **OpenTofu**: OpenTofu inherits many of Terraform's security features like state encryption and secrets management. However, as an open source project, security updates and audits depend on community contributions. Advanced features like RBAC will require external tools or third-party integrations.

5.5 Integration with Other Tools and Services

Integration with other DevOps tools is key for a seamless infrastructure management workflow.

- **Terraform**: Terraform integrates easily with many different tools and platforms. It fits right into CI/CD pipelines, works well with configuration management tools like Ansible, and connects with monitoring solutions like Datadog and Prometheus. Terraform also has built-in support for its cloud-based services, Terraform Cloud and Terraform Enterprise, which offer extra features for large-scale organizations. (HashiCorp, n.d., Terraform pricing).
- **OpenTofu**: Since OpenTofu is based on Terraform, it can integrate with the same tools and services, using the same providers and modules. While it lacks some of the advanced features that come with Terraform's enterprise offerings, organizations can use third-party tools to fill those gaps and still benefit from a robust integration ecosystem.

6 Migration Process

Migrating from Terraform to OpenTofu comes with its own set of technical and organizational challenges. In this chapter, a detailed process will be presented to ensure a smooth transition, covering everything from planning and script conversion to testing, tools, and backup measures.

6.1 Planning and Migration

Effective planning is vital for a successful migration. Organizations must take stock of their existing infrastructure and configurations before making the switch from Terraform to OpenTofu. This involves assessing current setups, pinpointing dependencies, and evaluating potential risks.

6.1.1 Assessing Existing Terraform Configuration

The first step in the migration journey is to review the current Terraform configurations thoroughly. It is important to consider the following aspects.

- **Infrastructure Complexity**: An evaluation should be conducted to assess the complexity of the current infrastructure. This includes understanding the number of resources in use, the types of providers (e.g., AWS, Azure), and any custom modules that are currently being utilized.
- **State Management**: An examination of how Terraform state files are being managed should be undertaken. This includes determining whether they stored locally, in AWS S3, or in Terraform Cloud. Knowing how the state is currently stored will guide the transition to OpenTofu.
- **Modules and Custom Scripts**: An inventory of any custom Terraform modules or scripts should be taken. Proper documentation and access to

these modules should be ensured, as they may require migration or rewriting for compatibility with OpenTofu.

6.1.2 Identifying Dependencies and Potential Issues

Identifying dependencies that may complicate the migration is essential. Organizations should assess the following:

- **Third-Party Integrations**: Any tools or services integrated with Terraform, such as CI/CD pipelines or monitoring tools, should be reviewed to verify their compatibility with OpenTofu.
- **Version Constraints**: The versions of Terraform providers and modules in use should be checked. While OpenTofu is compatible with Terraform providers, some might need closer examination to ensure they work seamlessly during migration.
- **Custom Resources**: If the organization relies on custom Terraform providers or unique workflows, evaluate whether these can be easily transferred to OpenTofu or if modifications will be necessary.

6.2 Step-by-Step Migration Guide

Migrating from Terraform to OpenTofu must be executed with care to avoid any disruptions. The migration process begins with installing OpenTofu, followed by renaming files and directories and updating provider blocks. Additionally, modifications to automation scripts may be required to accommodate the new tool.

Next, a staging environment is set up where dry runs are performed using OpenTofu to identify any potential issues before applying changes. Once confirmed, changes are applied incrementally, and the environment is closely monitored for any issues.

6.2.1 Converting Terraform Scripts to OpenTofu Scripts

Converting Terraform scripts to OpenTofu involves a structured and systematic process to ensure a smooth migration while minimizing potential issues.

The steps illustrated in Figure 4 are critical in ensuring that the migration process aligns with OpenTofu's structure.

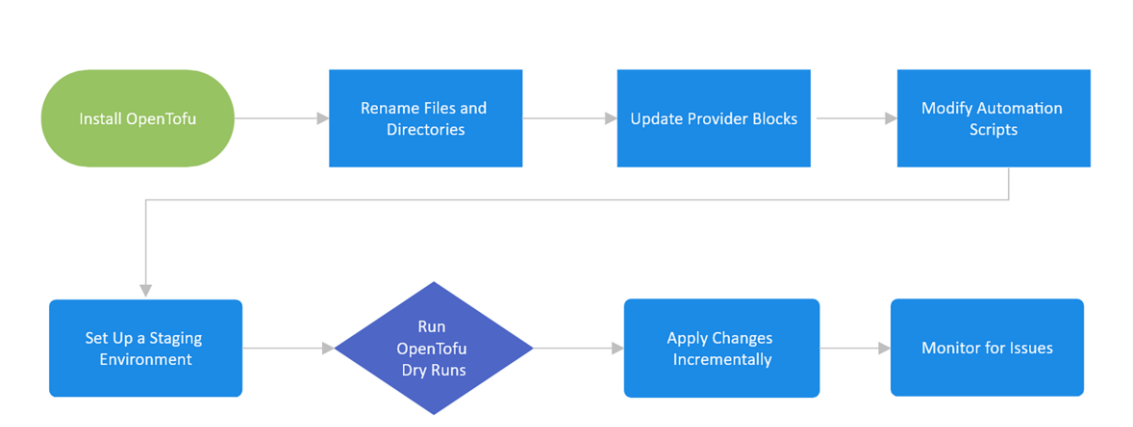

Figure 4. Conversion Process Diagram (Faezi, 2024)

Since OpenTofu is essentially a fork of Terraform, converting scripts is a relatively straightforward process. The syntax and structure remain largely unchanged. Here are the key steps:

1. **Installing OpenTofu**: Start by installing OpenTofu and ensure that the version is compatible with the existing infrastructure.
2. **Rename Files and Directories**: Replace terraform with tofu in the commands used in configuration files and scripts (e.g., terraform plan becomes tofu plan). Similarly, update module references if needed, especially for custom modules.
3. **Update Provider Blocks**: Confirm that any provider configurations are still valid. Usually, this requires minimal changes, as OpenTofu supports the same providers.
4. **Modify Automation Scripts**: Adjust any CI/CD pipelines or automation scripts to use OpenTofu commands instead of Terraform. This could

involve altering configuration files, deployment scripts, or workflows linked to version control systems.

6.2.2 Testing and Validation

Testing is a critical part of ensuring that the infrastructure operates as expected after migration. Here are some steps to follow:

- **Set Up a Staging Environment**: Before migrating the production environment, test the new OpenTofu configurations in a staging or development environment.
- **Run OpenTofu Dry Runs**: Use the command "tofu plan" to validate the changes. This command allows comparing the current state with the intended OpenTofu configuration without making any changes, helping identify potential issues.
- **Apply Changes Incrementally**: Instead of migrating the entire infrastructure at once, apply changes gradually to reduce risks. For example, consider moving specific components, such as networks or compute instances, one at a time.
- **Monitor for Issues**: After applying changes with "tofu apply," keep an eye on the infrastructure for any unexpected behaviours or issues. This includes verifying network configurations, application performance, and reviewing monitoring logs.

6.3 Tools and Best Practices for Migration

Successful migration of IaC configurations relies on effective tools and adherence to best practices to minimize disruptions.

6.3.1 Tools for Migration

Several tools can streamline the migration process:

- **OpenTofu CLI**: The OpenTofu command-line interface is the main tool for managing infrastructure. Since it supports the same commands as Terraform, this simplifies the migration process.
- **Version Control Systems**: Git or other version control systems are used to track changes during the migration. Clear version histories are maintained to enable easy rollback in case any issues arise.
- **CI/CD Pipelines**: Existing CI/CD pipelines are reviewed to ensure compatibility with OpenTofu withy modifications made to configuration files and scripts as needed. Tools like Jenkins, GitLab CI, or GitHub Actions can help automate testing and applying OpenTofu configurations.

6.3.2 Best Practices for Migration

- **Document the Process**: Detailed documentation of each step in the migration process is maintained. This includes records of any changes made to provider configurations, scripts, or custom modules.
- **Engage Stakeholders**: Key stakeholders, including the DevOps team, security personnel, and infrastructure owners, are involved to ensure that every aspect of the infrastructure is accounted for and verified during the migration.
- **Modularize Configuration**: Large Terraform configurations are divided into smaller, modular components if this has not already been done. This approach simplifies migration and testing of individual parts of the infrastructure.

6.4 Rollback and Contingency Planning

Every migration carries risks, so it is crucial to have plans in place for rollback scenarios and contingencies to minimize disruptions to infrastructure services.

6.4.1 Rollback Strategy

In the event that a migration encounters issues, having a solid rollback strategy is essential.

- **Backup State Files**: Before starting the migration, create backups of the Terraform state files. This step ensures that you can revert to Terraform if necessary, without losing any infrastructure configurations.
- **Parallel Environments**: If feasible, run parallel environments for both Terraform and OpenTofu. This strategy reduces risk by allowing comparisons between the two setups before fully transitioning to OpenTofu.
- **Revert Configurations**: If problems occur after applying OpenTofu configurations, revert by switching back to the original Terraform configurations and state files. Using a version control system can automate this restoration process.

6.4.2 Contingency Planning

Alongside rollback strategies, it is important to develop contingency plans to manage potential challenges during migration:

- **Downtime Planning**: For critical infrastructure, ensure there is a strategy in place to manage possible downtime. This could involve scheduling maintenance windows, notifying stakeholders, and preparing backup services.
- **Team Readiness**: Make sure the DevOps and IT teams are prepared to respond to issues quickly. Assign clear roles, establish a communication plan, and create protocols for troubleshooting and resolution.

7 Conclusion and Recommendations

This chapter summarizes the key findings of the research, provides answers to the research questions, discusses the practical implications of the study, and offers recommendations for practitioners and suggestions for future research.

7.1 Summary of Findings

The migration from Terraform to OpenTofu reveals a landscape rich with both similarities and differences. Key findings include:

- **Features and Usability**: Both Terraform and OpenTofu share a similar foundational architecture and syntax, which facilitates a smoother transition for users.
- **Migration Challenges**: The study identified several possible challenges associated with migration, including dependency management, compatibility of existing integrations, and the need for comprehensive testing. Organizations must recognize these challenges to effectively navigate the transition.

7.2 Answer to Research Questions

The comparison between Terraform and Opentofu reveals both key differences and similarities in terms of features and usability. Opentofu offers enhanced support for community-driven modules and specific performance optimizations (OpenTofu, 2024, OpenTofu 1.8).

At the same time, similarities in syntax and structure enable users to transition between the two tools with relative ease. However, migrating from Terraform to Opentofu comes with its own set of challenges such as managing

dependencies, ensuring compatibility with existing tools, and testing new configurations.

These challenges can be addressed through planning, stakeholder engagement, and a systematic approach to migration. To ensure a successful migration process, best practices include maintaining comprehensive documentation, adopting incremental migration strategies, promoting clear and effective communication among stakeholders, and performing continuous testing throughout the transition.

7.3 Practical Implications of the Study

The findings of this study offer valuable insights for organizations contemplating a migration from Terraform to OpenTofu. Understanding the similarities and differences between the two platforms equips decision-makers with the knowledge to make informed choices. By addressing the identified challenges and following best practices, organizations can enhance their infrastructure management capabilities and ensure a smoother transition.

7.4 Recommendations for Practitioners

Based on the findings, the following recommendations are made for practitioners involved in the migration process:

- **Conduct a Comprehensive Assessment**: Before initiating the migration, organizations should assess their existing Terraform configurations and identify potential risks and dependencies.
- **Engage Stakeholders Early**: Involving key stakeholders throughout the migration process ensures that all aspects of the infrastructure are considered, fostering collaboration and reducing the likelihood of oversights.

- **Implement Incremental Migration**: Practitioners should adopt an incremental approach to migration, allowing for gradual adjustments and testing, which can help mitigate risks and facilitate smoother transitions.
- **Maintain thorough Documentation**: Keeping detailed records of the migration process, including configurations, scripts, and any challenges faced, will provide a valuable resource for future reference and troubleshooting.

7.5 Suggestions for Future Research

Future research can build on this study by exploring several avenues:

- **Performance Analysis between the IAC tools**: Future research could involve hands-on studies that carefully examine how Terraform and OpenTofu perform in different situations. By doing this, researchers can uncover any important differences in how quickly each tool operates, how much resources they use, and how efficiently they manage infrastructure as code.
- **Development of Migration Frameworks**: Future studies could focus on developing frameworks or tools designed to assist organizations in navigating the migration process, addressing the complexities identified in this research.
- **Case Studies of Successful Migrations**: Documenting detailed case studies of organizations that have successfully migrated to OpenTofu can yield best practices and lessons learned that benefit the wider community.

References

Dadgar, A. (2023) HashiCorp adopts Business Source License. Available at: https://www.hashicorp.com/blog/hashicorp-adopts-business-source-license [Accessed: 20 August 2024].

Faezi, R. (2024) Terraform overview [unpublished diagram]. School of ICT, Metropolia University of Applied Sciences.

Faezi, R. (2024) Conversion Process Diagram [unpublished diagram]. School of ICT, Metropolia University of Applied Sciences.

GitHub. (2024) OpenTofu Registry. Available at: https://github.com/opentofu/registry/tree/main/modules [Accessed: 11 September 2024].

HashiCorp. (2023) HashiCorp Licensing FAQ. Available at: https://www.hashicorp.com/license-faq [Accessed: 29 September 2024].

HashiCorp. (n.d.) Terraform pricing. Available at: https://www.hashicorp.com/products/terraform/pricing [Accessed: 13 October 2024].

Michalowski, M. (2024) Benefits and Best Practices for Infrastructure as Code. Available at: https://devops.com/benefits-and-best-practices-for-infrastructure-as-code/ [Accessed: 20 August 2024].

OpenTofu. (2024) OpenTofu 1.8.0. Available at: https://opentofu.org/blog/opentofu-1-8-0/ [Accessed: 28 September 2024].

OpenTofu. (n.d.-a) OpenTofu Language Documentation: Providers section. Available at: https://opentofu.org/docs/language/ [Accessed: 22 August 2024].

OpenTofu. (n.d.-b) OpenTofu Language Documentation: Resource section. Available at: https://opentofu.org/docs/language/ [Accessed: 22 August 2024].

OpenTofu. (n.d.-c) OpenTofu Language Documentation: State management section. Available at: https://opentofu.org/docs/language/ [Accessed: 22 August 2024].

OpenTofu. (n.d.-d) OpenTofu Language Documentation: Modules section. Available at: https://opentofu.org/docs/language/modules [Accessed: 22 August 2024].

OpenTofu. (n.d.) Command Line Interface: CLI Commands. Available at: https://opentofu.org/docs/v1.6/cli/commands/ [Accessed: 22 August 2024].

OpenTofu. (n.d.) Manifesto. Available at: https://opentofu.org/manifesto/ [Accessed: 20 August 2024].

OpenTofu. (n.d.) What Are the Differences between OpenTofu and Terraform? Available at: https://opentofu.org/faq/#opentofu-terraform-differences [Accessed: 20 August 2024].

Sanders, J. (2023) Terraform Community Breaks Ground on Open Source Alternative. Available at: https://www.ccsinsight.com/blog/terraform-community-breaks-ground-on-open-source-alternative/ [Accessed: 20 August 2024].

Terraform. (n.d.) Manage resources in Terraform state. Available at: https://developer.hashicorp.com/terraform/tutorials/state/state-cli [Accessed: 21 August 2024].

Terraform. (n.d.) Modules overview state. Available at: https://developer.hashicorp.com/terraform/tutorials/modules/module [Accessed: 21 August 2024].

Terraform. (n.d.) Support for Terraform Enterprise. Available at: https://developer.hashicorp.com/terraform/enterprise/deploy/troubleshoot/contact-support [Accessed: 28 September 2024].

Terraform. (n.d.-a) Providers. Available at: https://developer.hashicorp.com/terraform/language/resources/syntax [Accessed: 21 August 2024].

Terraform. (n.d.-b) Resource Blocks. Available at: https://developer.hashicorp.com/terraform/language/resources/syntax [Accessed: 21 August 2024].

Terraform. (n.d.) What is Infrastructure as Code with Terraform? Available at: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code [Accessed: 09 October 2024].

Vivekanand, R (2023) Terraform Architecture Overview – Structure and Workflow. Available at: https://spacelift.io/blog/terraform-architecture [Accessed: 10 October 2024].

Appendix 1. Terraform Providers

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_s3_bucket" "example_bucket" {
  bucket = "example-bucket-name-12345"
  acl    = "private"
}
```

Appendix 2. Terraform Resources

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"  # Amazon Machine Image (AMI) ID
  instance_type = "t2.micro"               # Instance type

  tags = {
    Name = "ExampleInstance"
  }
}
```

Appendix 3. OpenTofu Providers

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

provider "aws" {
  project = "acme-app"
  region  = "us-central1"
}
```

Appendix 4. OpenTofu Resources

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```