International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

Infrastructure as Code (IaC) Evolution - A Comparative Study of Terraform and AWS Cloud Formation

Tapan Kumar Rath[1], Sibaram Prasad Panda[2]

Email: *tapankumarrath001[at]gmail.com*

Email: *spsiba07[at]gmail.com*

Abstract: ***Infrastructure management is one of the most important tasks in a cloud computing environment. Cloud infrastructures provide dynamic, distributed, and large-scale computing resources to help meet service scalability, availability, and performance requirements. Cloud providers offer their customers infrastructure resources that are managed and made available to users privately or publicly, charging based on consumption. The Infrastructure as a Service model provides virtualized computing resources stored in data centers and delivered on-demand using self-service capabilities, with provisioning options based on pre-allocated virtual machine images containing a guest operating system and any application software above that layer. Users who run their applications on the IaaS model are responsible for managing their operating systems and runtime software environments.***

Keywords: Hashicorp, Terraform, AWS CloudFormation, CI/CD. Infrastructure as Code

1. Introduction

Currently, many companies have adopted cloud computing, but most of them treat it merely as an extension of their on-premise infrastructures. In this approach, cloud computing serves as an external data center for hosting data and applications; organizations do not leverage the full potential of this powerful technology. We discuss Infrastructure as Code, a concept that enables cloud services to be a core part of businesses' everyday operations. IaC is based on the concept of treating infrastructure in a similar way to how software developers treat applications. In infrastructure provisioning, infrastructure code declares the needs of a specific application, enabling the automation of infrastructure provisioning and management. This Infrastructure as a Code declarative model enables management processes to adopt best practice software development life cycle techniques.

2. Background of Infrastructure as Code

Infrastructure is one of the first domains that companies migrated to the Cloud. Beginning with the creation of networking equipment, private cloud users started to virtualize their servers, later creating everything else required to operate their business issues, such as storage, monitoring, security devices, domains, test environments, and so on. Those cloud resources were required to be created and configured at the right time, with the required attributes and linked to each other correctly. Several kinds of SDKs were created to facilitate these tasks. However, it was not an easy task. Apparently simple environments, with only a few dozen resources, had to be customized using several SDK calls combined in some programming language. It would be around a decade later that a new efficient way of managing cloud resources appeared. The term Infrastructure as Code was coined, and Templates were proposed as a means to define cloud resources in a single place, properly defined as per specifications, and in a declarative manner.

3. Overview of Terraform

HashiCorp Terraform is an open-source IaC tool written in Go, released in 2014 by HashiCorp. Unlike existing IaC tools, Terraform is not tied to the attribute model of resources available for the specific cloud provider, resource types, and their property values. To enable extensibility and multi-cloud, Terraform incorporates pluggable provider-based architecture. This means that any cloud provider can build a Terraform provider and publish it in an open-source fashion. Terraform abstracts all servers, databases, networks, and security-related resources offered by the cloud provider into its resource-type and user-defined resource attribute structure/model. Cloud providers can extend the capabilities by their resource model without modifying the core Terraform code, which influences the wide adoption of Terraform by many enterprises.

3.1 History and Development

Terraform is a tool for managing infrastructure in a declarative way, which popularizes the term "Infrastructure as Code" (IaC). Terraform differs from most other provisioning tools in that, rather than treating the resources being provisioned as a series of mutable states, it is based on the concept of a graph of resources and manages the life cycle of that graph. This approach allows for more advanced features such as data flow management, intelligent renaming via "resource addressing", and parallel and dependency-aware provisioning of resources. Terraform, although largely oriented towards cloud resource provisioning, is very flexible and can manage other types of infrastructure, such as on-premises VMs and network devices.

Terraforms first release, version 0.1.0, was released on July 28, 2014. Terraform 0.2.0 was released in September 2014, when several AWS plugins were added, allowing Terraform to create AWS resources. The 0.3.0 version was launched the following month, and the 0.4.0 was released in November. In

February 2015, terraform 0.5.0 was released with new plugin documentation and several bug fixes. The latest version released in the 0.x series, Terraform 0.11.14 deprecated the use of legacy environment variable authentication, and reached end-of-life status on August 4, 2020.

3.2 Core Features

Terraform applies the concept of Infrastructure as Code by decoupling infrastructure specification from execution. Importantly, the abstraction provided is technology agnostic. Provisioners for practical cloud infrastructure platforms are implemented as plugins that implement a generic interface. Terraform uses an unmarshalled, domain-specific JSON, marking up the resource and data types, and their attributes, resource dependencies, and input arguments. The high-level configuration is provided in a DSL.

3.3 Use Cases

Terraform is primarily targeted at service providers like Google Cloud Platform and Azure. In addition, it is capable of supporting several non-cloud service providers, including Apcera, CloudStack, DigitalOcean, eBay, Fastly, GitHub, Heroku, Kubernetes, Mailgun, Pagoda, Openstack, OVH, and Qualys. The strength of Terraform resides in its ability to leverage larger plugin ecosystems and to support multi-cloud use cases. Tile38 and LINODE are currently two special services powered by Hashicorp using Terraform for enterprise cloud provider integration.

4. Overview of AWS CloudFormation

AWS CloudFormation offers a text-based interface that allows users to design, provision, and manage cloud-based infrastructures and higher-level services in the AWS cloud. Using source code to define their infrastructure, users free themselves from the inefficiency of manual GUI-based procedures. Users write, modify, and execute automation scripts to easily, reliably, and quickly create or configure many resources at once. CloudFormation can also create or modify resources following a defined process, such as enforcing a naming convention or ensuring security groups include specific ingress rules. Using CloudFormation code, source control and CI/CD processes integrate and automate security, compliance, and operational procedures for the entire AWS environment, increasing operational efficiency and reliability while reducing risks and costs.

4.1 History and Development

Infrastructure has grown increasingly important within Information Technology. Infrastructure refers to networks, servers, storage, and data centers that are responsible for hosting the most varied system solutions. In the past, companies built their infrastructure using physical components that were necessary for the services to function properly. However, these products came to possess certain limitations. They were exposed to greater chances of breakdown, in addition to requiring the craftsmanship of skilled professionals who dedicated a lot of time to doing maintenance and configuration tasks. With time passing, companies started adopting a digital version of these services hosted in Cloud Computing environments.

Cloud Computing provides an easy way to access, share, and alter virtual resources through the internet in a more optimized way. In a Cloud Computing environment, resources are placed in a shared pool that can be accessed when needed. Nevertheless, consumers started suffering from certain limitations related to costs.

4.2 Core Features

AWS CloudFormation is an infrastructure as code solution provided as part of the broader suite of services available with Amazon Web Services. It provides a way to describe and provision all of AWS's infrastructure resources in a cloud environment using text files, referred to as templates. These templates are text files, either in JSON or YAML format, that describe the infrastructure resources, their parameters, and their configuration that are needed to enable specific workloads or application deployment in the AWS ecosystem. With CloudFormation, users can use a single command line to provide many of AWS's cloud services, including storage, databases, networking, or compute resources on-demand. CloudFormation also integrates with many services in the AWS environment, including AWS IAM for access control and AWS S3 for storage of large template files.

CloudFormation automates the setup of stacks, which are groups of related infrastructure resources that are created, updated, and deleted as a unit. CloudFormation will not only create the specified stack but will also take care of the possible dependency connection order between resources. CloudFormation is a declarative IAC solution, which means that the user must declare what the end result should be. CloudFormation then uses its own engine to create the infrastructure resources to meet the specifications. CloudFormation provisions resources directly via API calls to AWS services using a mounting channel manager, which creates the request and returns the response from the API calls asynchronously. CloudFormation's template file can be mapped with any resource type supported by AWS CloudFormation. The mappings of resource properties and types do not have to be defined in CloudFormation. The API requests to provision the resource, however, have a defined schema.

4.3 Use Cases

AWS CloudFormation is a key component of the AWS ecosystem. You might imagine it doing everything from deploying a sandbox environment for testing and development to managing a multi-account, multi-region, cross-account configuration for a global web application. We explore a range of use cases for AWS CloudFormation below. One of the most common use cases for AWS CloudFormation is redeploying or updating a stack. Consider a case where a stack has already been created

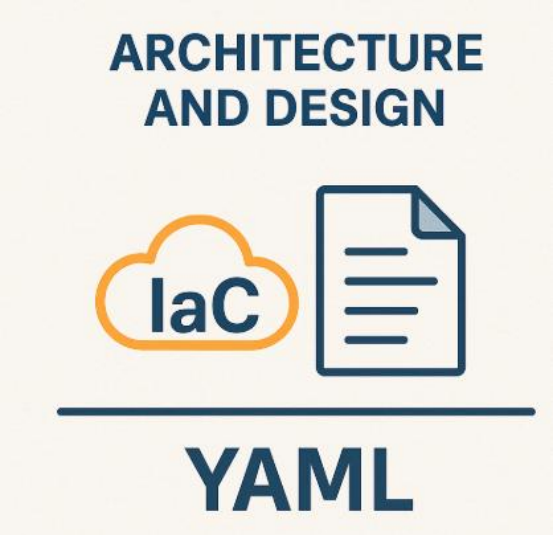
5. Comparative Analysis

In this section, we present a comparative analysis of Terraform and CloudFormation, with a focus on a number of aspects where we believe careful readers may be interested. Our comparative study is based on our knowledge of the systems.

We compare both IaC tools around these five themes: architecture and design, language and syntax, state management, ecosystem and community, and cost implications. First, let's address architecture and design.

5.1 Architecture and Design

Cloud infrastructure is composed of unmanaged or partially managed resources that are provisioned and configured without any form of automation. As cloud infrastructure scales up in size and increases in complexity, operating resources becomes an increasingly difficult task. Managing cloud infrastructure without automation becomes error-prone, resource-intensive, and difficult due to delayed response times. Infrastructure as Code is the approach of treating the entire cloud infrastructure as software code and YAML is the popular choice of developers to define this infrastructure. Infrastructure as Code uses code to define and maintain the lifecycle of cloud infrastructure using declarative or imperative statements. Declarative statements describe a desired state of the cloud infrastructure and imperative statements include provisions for how to reach that desired state. The advantage of the declarative format is that the existing resources do not need to be changed when the same code is executed after a period of time. IAC tools consume scripts in YAML format and are responsible for creating, managing, and updating resources. IAC tools come in two formats – those provided by the respective cloud providers and open-source ones that support multiple cloud platforms. Examples of first-party, proprietary, vendor-driven Infrastructure as Code tools are the various command line tools and SDKs provided by each public cloud. An open-source Infrastructure as Code tool called Terraform was created.



CloudFormation and Terraform are the most popular and widely used Infrastructure as Code tools. CloudFormation supports only AWS cloud services and abstracts and hides the complexity of managing security and account settings. Terraform is multi-platform, runs locally, is flexible, and contains many assets created by the community to support other cloud providers. Both tools use YAML files to define the cloud resources but have different languages, engines, and workflows for provisioning resources.

5.2 Language and Syntax

Both Terraform and AWS CloudFormation provide Domain-Specific Language (DSL) for defining workflows and can, therefore, provide from a syntactical point of view not only better readability but also could enable deployment without in-depth coding knowledge. However, they differ on how to approach the syntax design and capabilities. AWS announced the CloudFormation CLI which provides support for YAML and JSON DSLs, higher detection of dependencies, and the ability to add account-specific logic in workflows run by the companies. Terraform uses its own templating called HashiCorp Configuration Language (HCL) which is a declarative DSL that can be translated to JSON. While HCL has, as mentioned, similar capabilities as YAML, it has a simpler syntax targeting infrastructure workflows.

5.3 State Management

State is a central concept in the implementation of an IaC tool. In traditional computing and programming, state refers to values stored in memory and computing in-progress. In the case of IaC, state is about configuration of production infrastructure which is retrieved from the cloud provider interface. In an IaC operation, the system compares the actual and the desired states and identifies the components that need to be modified, such as added, deleted, or updated. The detection of differences occurs for every execution of the command.

5.4 Ecosystem and Community

Ecosystem and community are fundamental aspects of tooling functionality, with a particular emphasis on User-Defined Functions (UDFs) or modules in the design of tools such as

the two discussed in this paper. We are talking about a lot of extra functionality that is implemented, as open-source packages and libraries, given or shared free of charge, to increase the overall functionality available while reducing the overhead on the main developers. Furthermore, maintaining and improving this network of packages and solutions to edge cases becomes the responsibility of a large community rather than a small core of developers.

CloudFormation was launched with many templates defined to serve as example or guide for user-built functions. With templates being declared and not executable code, the options for parameterization are minimal, and extending the base modules is a cumbersome task at best. Terraform, on the other hand, was designed from the ground up with UDFs in mind. UDF support was made possible by combining a domain-specific language with a compiler capable of serialization for imports. It is a readable and mutable DSL with support for variables and primitive data types, which facilitates package creation.

5.5 Cost Implications

The question of costs cannot be trivialized when advocating for an approach to work in the cloud. Initial efforts may not require significant funding.

6. Performance Evaluation

Cloud computing's emergence as a disruptive technology in IT has led organizations to adopt cloud infrastructures, which can be realized through Infrastructure as a Service (IaaS) strategy. Cloud providers offer a variety of services covering multiple domains, such as computing power, storage, and application deployment. These heterogeneous services constitute the service model of the IaaS strategy. When implementing the IaaS strategy, organizations need to develop and deploy an IT infrastructure that offers a service model with numerous instances. Cloud providers recommend that organizations invest in the deployment process of the IaaS strategy because it is usually time-consuming. When organizations effectively address the deployment phase of the IaaS strategy, they can take advantage of the scalability capabilities of cloud providers.

6.1 Deployment Speed

Infrastructure as Code (IaC) tools are used for automating IT infrastructure deployment, provisioning, and management. However, the speed of deployment is likely to be a major issue when using them too. Consequently, this section experimentally examines the two selected IaC tools, Terraform and AWS CloudFormation, to determine which one is faster for deploying IT infrastructure in a single and multi-cloud environment. Unlike the tools' authors, who chiefly suggest that their tools work well to speed up the deployment, we go a step further by providing indepth data-driven discussion of our results not only to quantify the total deployment speed of each tool in different environments but also to estimate systematically the contribution of each needed step to the overall deployment speed. Our study purposely covers two different environments to emulate real-world application requirements. Our experimental results show that while Terraform is consistently an order of magnitude faster than CloudFormation for deploying IT infrastructure in a single-cloud environment, such a speed advantage is more subdued and can be particularly reversible in a multi-cloud environment.

6.2 Scalability

In this section, we discuss our findings on the scalability of Terraform and CloudFormation. Scalability is a crucial aspect when it comes to deploying large infrastructures. Organizations that are heavily dependent on cloud services often manage the entire lifecycle of hundreds or thousands of resources. Consequently, tooling becomes a bottleneck. Most entities would select the tool which could help execute operations faster. Scalability determines how well a system can enlarge to cater for more use, the expansion of a system's capacity accompanied with its decrease in performance and the speed ratios of larger systems. This paradigm operates by observing that the performance of a system changes as the workload changes in a specific manner based upon the specific underlying characteristics of the workload.

To conduct these experiments, deployment is made on a machine with Intel Core i7-4790 CPU @ 3.60GHz and 64GB RAM. For the testing phases, we created samples of different sizes by duplicating the necessary resources to be deployed from the YML manifests specified before. We then imported the respective modules to CloudFormation and Terraform and launched the stack and plan apply. To calculate the resources used, we averaged the execution times of five consecutive runs of each of Terraform and CloudFormation stack launches. For our load testing, we executed the tests on stacks with sizes ranging from 1,700 to 60,000 resources. As it would not be possible to run these tests for more than 50,000 resources using CloudFormation due to AWS quotas, we made use of approximately 600 modules to deploy a test with 100,000 CloudFormation resources.

6.3 Reliability

Deploying cloud infrastructures in a reproducible manner by using IaC is a serious effort. But what happens if the IaC tool itself is not reliable? Underlying failures and bugs in the IaC may greatly disrupt the deployment and management of underlying cloud infrastructures. We report some relevant facts that show that one tool is much more prone to errors and bugs than another.

7. Security Considerations

As with any interface to technology, security must be a paramount concern. IaC has the potential to combine the convenience of a tooling interface, with the power over significant automation and capabilities to allow extensive manipulation of cloud resources. Consequently, IaC could allow for the exposure of sensitive capabilities if not secured properly, potentially damaging any privacy and compliance guarantees important to organizations using IaC.

Since IaC is a technology for managing deployment templates, the data processed includes the conditions of the environment and how resources will be configured and

associated together. Managing this data represents a significant hurdle to IaC, requiring special consideration into pipelines and role designs, policies, and encryption/ decryption capabilities.

7.1 Access Control

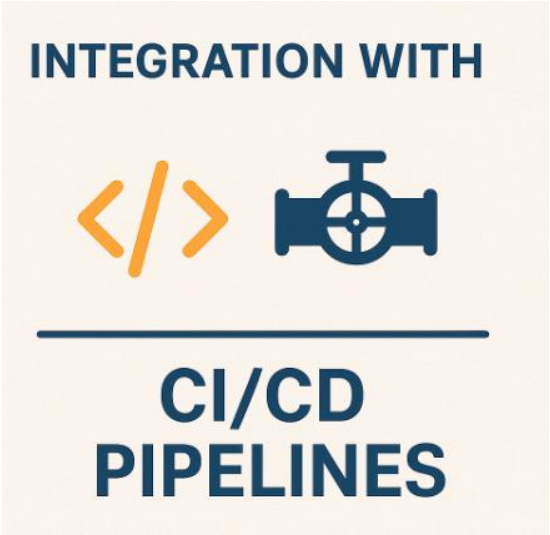
Security is a point of concern for users, as infrastructure defines the ecosystem in which services that users consume usually run. Any unwanted modification can compromise the integrity of the hosted services. Furthermore, IaC gives the possibility of creating a large set of resources within a single command. A problematic script, either intentional or by mistake, can lead to the creation of many resources, consuming quota limits and, even worse, generating loads of expenses for the user. Therefore, restricting access to IaC files, using granular permissions based on the user/application profile, and restricting the environment where scripts can be run are key aspects of security. Restricting bulk operations for certain users with a single policy is a way to reduce damage. For instance, a role that enables the creation of instances, of type t2. micro, in only one or some developed environments, could be created and granted just to developers that need it. All other resources needed could be created in the allowed environment without any policy that limits their usage.

7.2 Data Protection

Although the importance of data protection is widely known and should be one of the primary bases for system and infrastructure design, solutions focusing on data protection are only being developed slowly. In either tool, a stack can be applied to encrypt any secrets used, but hard work must be invested in protecting other resources. Therefore, the resources representing the data encryption itself are expected to be manually created with the highest possible attention to detail, and only procedural-related resources and data flows should be controlled with tools. Both tools have some primitive capabilities for infrastructure protection against unintended access but only for certain keys.

8. Integration with CI/CD Pipelines

The need for agile development has led to rapid change in best practice provisioning, configuration management, continuous integration, and continuous deployment. Infrastructure is a key enabling factor in DevOps since the goal is to increase the frequency of change and minimize the risk involved. Applications derive risk from two sources: flaws in the application and flaws in the infrastructure that the application depends upon. Code analysis, testing frameworks, staging deployment environments, shadow traffic, and automatic rollbacks are all measures that can be taken to increase the safety of deployments. The other aspect is cost. For a cloud-hosted service, Infrastructure as a Service provisioning converts the capital cost of increasing infrastructure capacity into real-time variable expenditure. But actual demand for services is sporadic, so using tools for automatic scaling is critical, and driving this use through automated provisioning templates is key to enabling speed and safety while minimizing spam on cloud providers. It is essential to ensure that a failure in the infrastructure will trigger an alert early enough to take action that mitigates damage. Terraform is designed to manage infrastructure as a part of a larger software project, so it naturally fits into CI/CD processes. Creating a cloud-hosting service typically involves components that are developed separately but need to be merged



8.1 Terraform in CI/CD

The idea of Infrastructure as Code (IaC) has solved so many problems faced in using any cloud infrastructure that people began delegating more and more within their pipelines usually provided by Continuous Integration and Continuous Deployment (CI/CD) tools. Terraform is a candidate for providing these delegations with its available libraries and decent template support.

Plan States Out of Reach: It is best to avoid keeping. tfstate files in the same repository as the code that is responsible for its updates. Developers with write access to the repository may be able to change its contents and record invalid data or even delete it altogether without running a plan step.

'Prevent*' locks after several seconds: Running long plan/destroy tasks may lead to concurrency problems, but this may be tolerated on smaller projects due to Terraform's inherent lack of support for multi-user usage and operations. This can either be avoided by using only one CI/CD agent at once or by using Terraform Cloud.

The right permissions for CI/CD: A common practice is to make the CI/CD server run in its own service account with only enough permissions to run the planned commands and not everything with elevated privileges. Combining this with access tokens associated to the CI/CD account with the same methods used for any other credentials would solve this.

8.2 AWS CloudFormation in CI/CD

With advantage of being the native solution of AWS, AWS CloudFormation benefits from good integration with other AWS services. AWS CloudFormation interacts with several services to implement cloud resources, a review of AWS CloudFormation raises several aspects to be analyzed. For example, Billing and Cost Management service is used to track estimated cost of CloudFormation stacks and their related resources, CloudWatch service for service metrics and related alerts, CloudWatch Events service to identify stack related events, CloudTrail for event logging, Organization for service control policies, and CloudFormation Drift Detection for validating stacks and preventing drifts caused by direct modifications in stack related resources.

9. Best Practices

All software development carries some form of risk. The practice of IaC facilitates manageable systems as a form of documentation. Since IaC consists of code files, these files can be stored in a version control system. Furthermore, version control also allows multiple individuals to write IaC at the same time and merge their work into a single codebase when done. This process of merging should be strongly outlined in the procedures regarding inputting IaC onto the version control system. These decisions include what approach to use for reconciliation, who has general backing permissions, and how widely these permissions are assigned. The same IaC guidance for regular software development can apply here. Use comments in the code describing why it is done in a specific manner instead of merely what it is doing.

9.1 Version Control

Version control is a cornerstone of delegate programming and best practices ensure that services can be viewed and updated in a safe way. As we expect teams to split responsibilities and share core components, team trust is enforced by controlling code changes. Code sharing platforms are ubiquitous, and most organizations encourage code sharing for reusable services even when management overhead may be high. The collaboration model tries to address these concerns by laying down conventions for branching and issuing pull requests, whilst enables code management for more complex use cases through its Individual Workspaces feature.

9.2 Modular Design

Beginning with simple applications, the user of the IaC tool usually creates and maintains monolithic templates, which after years have hundreds of thousands of lines of code. Problems with the creation and maintenance of those types of files have led to the emergence of the partial provisioning concept. Although that concept is somehow implicit in the documentation of both tools, none of them suggested an architecture decomposed into multiple services, each of them described by its own configuration or template file. Analyzing the guidelines, we can compare partial provisioning features made available by both tools: service support modules, partial provisioning abstraction, and state separation are features supported by both tools. However, only one tool provides deployment and plan separation, and service dependencies separation.

10. Future Trends in IaC

10.1 Emerging Tools and Technologies

The advent of Infrastructure as Code (IaC) has revolutionized the way organizations manage infrastructure; nevertheless, IaC is still a comparatively new concept, which is manifested in that beyond Terraform, a limited number of tools provide support for key capabilities, such as multi-cloud application management and collaborative capabilities. In the near future, we expect to see further development of the ecosystem surrounding Terraform, as well as development of new IaC conceptualizations that offer alternative solutions to common challenges facing organizations that have adopted IaC. Moreover, we anticipate that there will be increased demand for capabilities such as organizational policy as code tooling and workflow and productivity tooling around policy as code initiatives. These capabilities have so far served only niche audiences, which has resulted in a limited number of solution vendors in the market today. We also expect to see an increasing number of tools that leverage smart algorithms to increase automation of IaC process lifecycles.
10.2. Industry Adoption

10.2 Emerging Tools and Technologies

Even though the tools and technologies that lay the groundwork for IAC have existed for a while, the focus on IAC tools and technologies has grown during the past five years. The relationship among software engineering, development, and operation teams known as DevOps has created increased interest in managing infrastructure through code. As a result of the predictable and repeatable methods IAC enables, tech giants have created large language models for IAC management. A company is using open source code to augment its IAC efforts with LLMs. A startup is developing tools for what it calls taxonomic infrastructure, which automatically writes IAC code for infrastructure taxpayers can understand. Other startups are also writing IAC code to help businesses reuse code.

10.3 Industry Adoption

Despite CloudFormation being a truly mature tool, developed and enhanced exclusively for AWS, Terraform is highly popular in the automation space, being used and adopted by many enterprises. Apart from providing automation support for multiple clouds and other tools, Terraform also provides better usability, flexibility, and simplicity via modules and its state management capabilities, especially in multimodal and multivendor implementations. Over 1000 Terraform modules have been created by various contributors in the Terraform Registry. The most popular modules contain several AWS services like VPCs, EC2, RDS, IAM, and EKS. Industry

reports show that the demand for Terraform skills in the job market is huge, and the trend is increasing. In 2020, over 10,000 job postings were published with "Terraform" listed among the skills, and this number increases yearly. In another survey, 65% of the participants stated that their companies are using Terraform or plan to use it in the coming year or two. This inclination has a clear basis regarding the scope, advantages, and flexibility Terraform provides. These attributes align with the modern state of IT infrastructure management, which requires both agility and stability. In the last few years, Terraform became popular and the go-to Infrastructure as Code tool for many enterprises and agencies around the world from different industries. Companies use Infrastructure as Code Terraform to manage their cloud infrastructures on various platforms.

11. Case Studies

Infrastructure as Code (IaC) is revolutionizing the ways in which Cloud Computing resources are provisioned and managed. Terraform and AWS CloudFormation are two of the most popular Infrastructure as Code (IaC) tools. In the previous sections, we compared Terraform and AWS CloudFormation in terms of various aspects, such as Architecture, Language, Configuration, Design, Benefits, Drawbacks, Use Cases, and Security. Based on this analysis, we validated our claims through two case studies: a Terraform and AWS CloudFormation implementation of a Web Application in different IaaS environments.

11.1 Case Study 1: Terraform Implementation

To illustrate the differences between the two tools in comparison, we have selected a simple network architecture deployed on service providers using both tools. The objective of this study is to point out the differentiating points in their architecture and none of them is better than the other. We simply show the advantages and drawbacks of each tool. While all Content Delivery Network (CDN) providers offer standard features to deploy services (some of which are already being default), it is the direction towards which they have chosen to pursue their development that make their solutions distinct. Thus, this study was carried out to observe and analyze the similarities and the differences in both tools mentioned above. We compared and reviewed the configurations, the approaches used to build the available server and configuration types and which tools allow easier configuration management.

.

11.2 Case Study 2: AWS CloudFormation Implementation

We chose to implement the same IaC solution for our second case study using a built-in IaC tool and service. This tool allows users to declare and provision infrastructure, described in a template file, through a push mechanism. The daemon powerfully orchestrates the dynamic provisioning of data, relies on several APIs, and ensures that the current infrastructure state always matches the declared configuration template.

The key features of this tool are as follows: It supports all services offered by the platform, the largest cloud infrastructure platform. The syntax chosen implicitly adopts the data model and sets of API calls and parameters used by each service. Data can be expressed in any programming language that can issue API calls; these are usually helper wrapper libraries. The source of the template can be any source. The tool implements a push-model mechanism through which customers declare and push templates for processing.

12. User Experiences and Feedback

The evolution of the cloud ecosystem, empowering users to easily manage Infrastructure as Code (IaC), such as Infrastructure provisioning and orchestration, is the cornerstone of diverse collaborative development models, along with the advent of the share economy. In tracing the evolvement of tools and surveying the industry for feedback and recommendations, we discovered that several well-recognized companies utilized both Terraform and CloudFormation for the core strength of each. User feedback resonates with our observations. Many users note having more favorable experiences with Terraform. In particular, for managing multi-cloud or hybrid cloud architecture, Terraform is the better choice. Terraform started as a primary multi-cloud solution and has matured in that space since. Users like how

13. Conclusion

In this work, we presented an in-depth discussion about Infrastructure as Code (IaC), an emerging practice that allows for the automation of creation and management of the infrastructure of a software solution. We presented a background on the subject detailing its goals, and why it has gained traction in the past years, besides discussing other important subjects, such as tools, languages, providers and anti-patterns. To help carry the burden of our many discussions, we drew the concepts presented in the form of a comprehensive ontology. With this background, we then presented our primary focus, the comparative study of the two most widely used IaC tools: Terraform and AWS CloudFormation. Our comparison focuses on three aspects: the expressed intent behind their design choices, their strategic decisions as far as limitations and extension mechanisms and their language structural features. Based on our comparison, we then provided an informed discussion on which tool would be the most suitable for 17 different scenarios, scenarios that we carefully crafted based on the conceptual implications raised in our comparison.

References

[1] Saavedra, N., Gonçalves, J., Henriques, M., F. Ferreira, J., & Mendes, A. (2023). Polyglot Code Smell Detection for Infrastructure as Code with GLITCH.
[2] Verdet, A., Hamdaqa, M., Da Silva, L., & Khomh, F. (2023). Exploring Security Practices in Infrastructure as Code: An Empirical Study.
[3] Howard, M. (2022). Terraform - Automating Infrastructure as a Service.
[4] Borovits, N., Kumara, I., Di Nucci, D., Krishnan, P., Dalla Palma, S., Palomba, F., A. Tamburri, D., & van den Heuvel, W. J. (2022).

[5] Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2018). Where Are The Gaps? A Systematic Mapping Study of Infrastructure as Code Research.
[6] Dorodko, S. & Spillner, J. (2019). Selective Java code transformation into AWS Lambda functions.
[7] Bhattacharjee, A., Barve, Y., Gokhale, A., & Kuroda, T. (2019). CloudCAMP: Automating Cloud Services Deployment and Management.
[8] Mikkelsen, A., Grønli, T. M., & Kazman, R. (2019). Immutable Infrastructure Calls for Immutable Architecture.
[9] Lourenço, P., Pedro Dias, J., Aguiar, A., & Sereno Ferreira, H. (2019). CloudCity: A Live Environment for the Management of Cloud Infrastructures.