Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2020, 7(6):279-284

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Automated Infrastructure Provisioning and Management with Infrastructure as Code (IaC) Tools

Mounika Kothapalli

Senior Application Developer at ADP
Email: moni.kothapalli@gmail.com

Abstract This paper reviews infrastructure as code tools and best practices with respect to the existing three major solution options: Terraform, Azure Resource Manager Template, and AWS CloudFormation. In this paper, the most important features, capabilities, and use cases of each tool are explained, and then a minute comparison with respect to each of them will be performed in view of parameters such as ease of use, flexibility, performance, cost implications, and integration capabilities. Best practices for version control, collaboration, and continuous integration/continuous deployment in IaC environments are also taken into consideration in the paper. With these objectives in mind, this research aims to provide insights useful for any organization plotted on the roadmap to either plan or improve their Infrastructure as Code practice for enhancing their capabilities in cloud infrastructure management. It concludes the discussion with the emerging trends in IaC and some practical recommendations while adopting these technologies for organizations.

Keywords Infrastructure as Code (IaC), Cloud Computing, Terraform, Azure Resource Manager (ARM) Templates, AWS CloudFormation, DevOps, Continuous Integration/Continuous Deployment (CI/CD), Version Control, Cloud Infrastructure Management, Automation, Best Practices.

1. Introduction

A. Overview of Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is the management and provision of computing infrastructure through machine-readable definition files, instead of physical hardware configuration. IaC is a key DevOps practice, bridging the gap between development and operations by automating infrastructure setup and management. This approach allows for consistent and repeatable configurations, reducing the potential for human error and enabling rapid scaling and deployment of applications [1].

IaC is important because it makes the process of deployment and infrastructure management automatic and easier. Treating infrastructure as code allows for consistency, repeatability, and scalability of cloud environments in organizations. This, therefore, resonates well with the DevOps practices of supporting fast iterative development cycles so common in modern software development [2].

The benefits of using IaC include increased efficiency and speed of deployments with reduced human errors. It also allows enhanced version control, change management, cost optimization through better resource management [2].

B. Purpose of the Research

This paper is aimed at providing a comprehensive review of the tools and practices of Infrastructure as Code, focusing on three of the most popular solutions: Terraform, Azure Resource Manager Templates, and AWS CloudFormation. The following features will be pursued in this study:

1. **Understanding different IaC Tools:** I cover the main features, capabilities, and use cases for Terraform, ARM Templates, and AWS CloudFormation. This analysis will be conducted to provide insight into the strengths and weaknesses of each of these tools [3].
2. **Comparing Terraform, ARM Templates, and AWS CloudFormation:** A detailed comparison of these tools will be carried out with respect to parameters or factors such as usability, flexibility, performance, cost implications, and integration capabilities. This comparison will help an organization to make an appropriate decision while choosing an IaC tool to implement its requirements.
3. **IaC—Version Control and Collaboration, Best Practices:** Here, the paper will be focused on the version control systems, techniques for collaboration, and best practices for the management of IaC code. This will equally include modularization, testing strategies, and continuous integration/continuous deployment pipelines for IaC.

Therefore, by studying these areas and the insights offered, the organizations could adopt or improve their IaC practices.

2. Understanding Infrastructure as Code Tools

A. Terraform

Terraform is an open-source tool for Infrastructure as Code developed by HashiCorp. This assists in the provision of the infrastructure resources across a number of cloud providers using HashiCorp Configuration Language, HCL, which is a declarative language [3]. The main features of the terraform tool include:

- It has declarative syntax to define infrastructure
- Plan and Apply workflow for controlled changes
- State management, which manages changes of resources
- A module system that provides for code reusability
- The provider ecosystem offers multi-cloud support
- The Terraform Registry shares and discovers modules.

In addition, Terraform supports a wide variety of cloud providers, like AWS, Azure, Google Cloud Platform, and many more. It would be useful in scenarios when it comes to deploying resources to multiple clouds, for large enterprises, and/or companies that are looking for provider-agnostic IaC solutions.

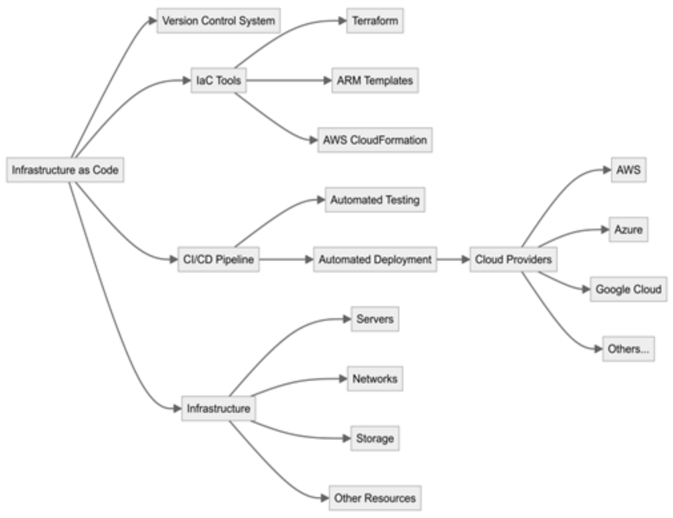

Figure 1: Overview of Infrastructure as Code

B. ARM Templates

ARM Templates are JSON-based files used for defining and deploying infrastructure resources in Microsoft Azure. They are native to the Azure ecosystem and hence very well integrated with Azure Resource Manager. The main features of ARM templates include:

- JSON-based declarative syntax

- Azure-native integration and optimizations
- Template functions for dynamic resource creation
- Nested and linked templates for modular designs
- Azure deployment modes: incremental and complete
- Integration with Azure Policy for compliance Specific use cases of managing Azure resources

ARM templates excel in pure Azure environments, as they can make a deep integration with Azure services. They are appropriate for an organization strongly invested in the Microsoft ecosystem, those having complex Azure deployments, or scenarios that would require fine-grained control over Azure-specific features [4].

C. AWS Cloud Formation

AWS CloudFormation is the native Infrastructure as Code service from Amazon, modeling and provisioning AWS resources through templates in JSON or YAML [5]. The key Features and Capabilities include:

- Writing templates in JSON or YAML format
- Resources can be managed by means of stacks and stack sets
- Change sets for previewing the changes in the infrastructure
- Drift detection identifies manual changes
- Modular designs by nested stacks.
- Integration with any AWS feature or service

CloudFormation remains the primary choice in AWS-centric environments, which are natively integrated with AWS services. It finds special utility among companies heavily invested in the AWS ecosystem, leading-edge AWS architectures, and particularly those making use of singular AWS capabilities [6].

Each of these IaC tools have their strengths and a few scenarios where each one is best suited. Terraform has its strength in its multi-cloud, ARM templates shine in Azure-native environments, and CloudFormation can be used best in AWS-centric infrastructures. That is, in many cases, the selection between them often depends on the particular cloud strategy of any given organization, existing investments, and required features.

3. Comparing Terraform, Arm Templates, and Aws Cloudformation

A. Ease of Use and Learning Curve

Terraform provides a command-line interface and has comprehensive documentation. ARM Templates use JSON syntax and integrate with Azure portal, while CloudFormation supports both JSON and YAML with integration in the AWS Management Console. The Azure portal and AWS management console offers graphical interface for managing and creating templates.

Terraform has a large, active community and extensive third-party resources. ARM Templates and CloudFormation have strong support within their respective ecosystems, but with somewhat smaller communities compared to Terraform [1].

B. Flexibility and Extensibility

Terraform provides high flexibility with its provider system and ability to create custom providers. ARM Templates offer Azure-specific customizations, while CloudFormation allows customizations through custom resources [1].

Support for various cloud services and integrations: Terraform excels in multi-cloud scenarios, supporting numerous providers. ARM Templates and CloudFormation are optimized for their respective cloud platforms but have limited cross-platform capabilities.

C. Performance and Efficiency

All three tools provide efficient deployment capabilities. Terraform does this with its state management, which enables faster incremental updates. ARM Templates and CloudFormation do the same with platform-specific optimizations [7]. Each tool provides mechanisms for resources optimization. Terraform's state management gives an evident view of the infrastructure, while ARM templates and CloudFormation are much more deeply integrated into the respective platform's Management features [3].

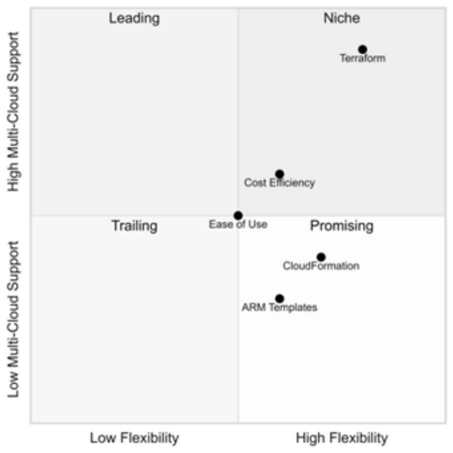

Figure 2: Comparing IaC tools: Terraform, ARM Templates, and CloudFormation

D. Cost Implications

Terraform is Open source, free to instantiate, with paid enterprise features. ARM Templates and CloudFormation are free services, while the cost is on the resources that will be provisioned [5].

All of the tools can aid in driving savings through automation and resource management, but incorrect usage of any one of them could result in runaway cloud resource costs [7].

E. Integration with other tools

Deep integrations into popular CI/CD systems are supported by all three tools. Although Terraform, in theory, provides flexibility due to its provider-agnostic nature. ARM Templates and CloudFormation open up through native integrations with the CI/CD services from their respective cloud platforms [6].

Terraform supports most of the monitoring tools, which are from different platforms. ARM Templates and CloudFormation have tight integration with the cloud platform's monitoring or management services. The quadrant chart in Fig. 2 shows Terraform as a leading tool with high flexibility and multi-cloud support. ARM Templates and CloudFormation are shown as more specialized tools with high flexibility but lower multi-cloud support.

Finally, though all these three tools are effective solutions for IaC, there are respective areas at which they shine in particular. On the strong points of Terraform are its multi-cloud features and huge community. Then there are the ARM Templates that come with deep Azure integration, while CloudFormation offers fuller functionality for AWS services. Most times, how one chooses between them often lies in an organization's particular cloud strategy and existing investments.

4. Best Pracitses for Version Control and Collaboration in Iac

A. Version Control Systems

Version control is crucial to IaC as it offers a means of tracing all changes done to a configuration, which allows for teamwork, and creates a record of how infrastructure has changed over time. Git is often used due to its distributed nature and branching capabilities. Organize your IaC code into logical modules [1]. Configuration should be separated from code. Apply branching strategies that mirror your deployment workflows. A well-organized repository should normally include directories regarding at least different environments, plus some core reusable modules and documentation. Following a clear naming convention and directory structure will make it easier for all members to find their way around and understand the codebase. Directories for modules, environments, and scripts could be created, from which the navigation will go on to make things easier and faster [3].

B. Collaboration and Code Review

Setting up a code review process allows for quality and uniformity of the code. It opens up an opportunity to have peers review changes before they are merged into the main branch. Thus, it facilitates the detection of any errors and thus establishes best practices, knowledge sharing. Use pull requests to suggest and review changes

[1]. Implement peer reviews such that changes are inspected by multiple members of your team before deployment. It doesn't only improve the quality of IaC but is also a learning exercise for the team members.

C. Modularization and Reusability

Designing reusable modules and templates is a best practice in the sense that it supports code re-use and consistency. Reusable modules may contain common configurations and may, therefore, be useful for a number of different projects, reducing duplication and consequently making maintenance easier [3]. For example, creating a module to set up a standard virtual machine deployment saves time and guarantees consistency between different environments. Modularization as showed in Fig. 3 improves maintainability by taking complex configurations down to manageable, re-usable components. This makes it easier to change and maintain the code since, once modified in one module, the change will then appear in all instances that use that module. Moreover, modularization allows scalability because teams can compose infrastructure configurations from well-defined building blocks.

D. Continuous Integration and Continuous Deployment (CI/CD)

CI/CD pipelines automate testing, validation, and deployment of infrastructure configurations for IaC. It enables the continuous integration of code changes and deploys them automatically to various environments, thus making sure that changes in infrastructure configuration can be applied uniformly and reliably. This will reduce manual effort and increase the speed of deployment. Some of the tools used in automating deployments for IaC include Jenkins, GitLab Ci, and GitHub Actions [8]. Best practices for automated deployment include clear definition of deployment stages, such as testing, staging, and production, automatic rollbacks in case of failures, and continuous monitoring for possible issues in the deployed infrastructure.

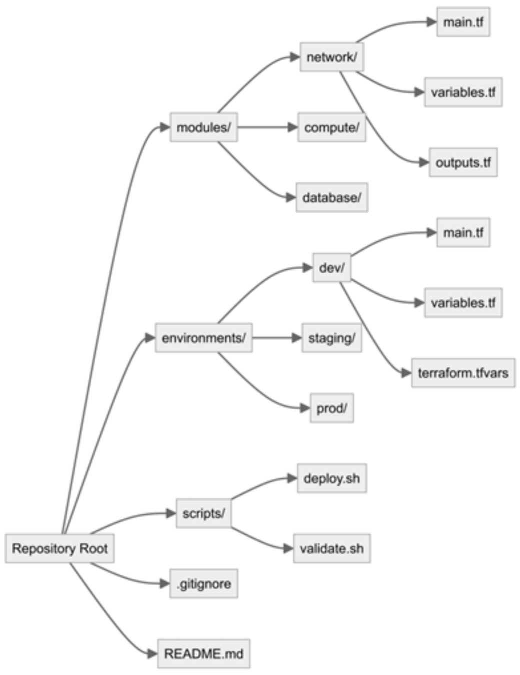

Figure 3: Well-organized IaC repository

E. Documentation and Knowledge Sharing

Thorough documentation is essential for an IaC codebase in terms of obtaining any understanding and making a contribution. In this case, it should focus on usage of modules and their purpose, configuration details, deployment processes, as well as details on how to perform these tasks. If the code is well documented, then it provides an easy onboarding experience for new members and offers reference points on troubleshooting and maintenance. Knowledge sharing across the team through scheduled meetings, an internal wiki, and collaborative effort while writing documents [1].

These best practices provide huge improvements in IaC processes within organizations, targeting more reliable, maintainable, and efficient infrastructure management. They also remain very close to DevOps principles through the improved collaboration between development and operations teams, as well as general quality and regularity in infrastructure deployments.

5. Conclusion

A. Summary

The comparison of Terraform, ARM Templates, and AWS CloudFormation reveals distinct advantages and limitations for each tool. Terraform excels in multi-cloud, as it provides great flexibility. The ARM Templates provide the deepest integration with Azure services, as well as AWS CloudFormation integrates well with the AWS ecosystem [1]. Basically, the choice of tool will heavily depend on the specific cloud strategy and previous investments an organization has made. Key best practices that come into play are version controls, code reviews, modularization of infrastructure code, automated testing and validation, integration with Continuous Integration/Continuous Deployment pipelines, and good documentation. All of them are close to the principles of DevOps and allow one to make infrastructure management more reliable and effective [8].

B. Future Trends

The field of IaC is rapidly evolving, with trends toward more automation, improved integration with container orchestration platforms, and more advanced testing and validation tools. There is also a greater focus on security and compliance automation within IaC workflows [6]. Key focuses for future development in cloud infrastructure management are visualized by doing betterment in automation and intelligence. AI and ML will be leveraged more to automate infrastructure provisioning, scaling, and optimization. Interoperability between a wide array of tools and platforms using IaC will make multi-cloud and hybrid cloud deployments easier to execute [7].

Final thoughts on the need for IaC in modern cloud infrastructure management are: Infrastructure as Code has become the keystone of modern management of cloud infrastructure. It gives organizations better efficiency, consistency, and reliability in running large and complex infrastructures. Treating infrastructure like code enables businesses to bring in the best practices from software development into their operations and manifested collaboration between the Development and Operations teams to finally offer value to their customers at a higher pace and more consistently [8].

References

[1]. Kief Morris, "Infrastructure as Code: Managing Servers in the Cloud," O'Reilly Media, Inc., 2016.

[2]. Nathaniel Felsen, "Effective DevOps with AWS: Implementing Continuous Delivery and DevOps," Packt Publishing, 2017.

[3]. Yevgeniy Brikman, "Terraform: Up & Running: Writing Infrastructure as Code," O'Reilly Media, Inc., 2017.

[4]. S. Sharma and A. Soni, "A comprehensive study on Azure Resource Manager," in 2016 International Conference on Computing, Communication and Automation (ICCCA), 2016, pp. 1470-1475.

[5]. N. Sabharwal and M. Wadhwa, "CloudFormation," in Automation through Chef Opscode: A Hands-on Approach to Chef, Berkeley, CA: Apress, 2014, pp. 71-88.

[6]. C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," IEEE Software, vol. 33, no. 3, pp. 94-100, 2016.

[7]. L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too," IEEE Software, vol. 32, no. 2, pp. 50-54, 2015.

[8]. G. Kim, J. Humble, P. Debois, and J. Willis, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press, 2016.