Proceedings of 2024 International Conference on Cognitive Robotics and Intelligent Systems (ICC - ROBINS 2024)
IEEE Xplore Part Number: CFP24UD1-ART; ISBN: 979-8-3503-7274-8

Streamlining Multi-Cloud Infrastructure Orchestration: Leveraging Terraform as a Battle-Tested Solution

2024 International Conference on Cognitive Robotics and Intelligent Systems (ICC - ROBINS) | 979-8-3503-7274-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICC-ROBINS60238.2024.10533995

Aniruddha Ghosh
Department of Networking and Communications
SRM Institute of Science and Technology
Kattankulathur, Chennai, Tamil Nadu, India
ag7434@srmist.edu.in

Sudhanshu Srivastava
Department of Networking and Communications
SRM Institute of Science and Technology
Kattankulathur, Chennai Tamil Nadu, India
ss1163@srmist.edu.in

P Supraja
Department of Networking and Communications
SRM Institute of Science and Technology
Kattankulathur, Chennai Tamil Nadu, India
Corresponding Author
suprajap@srmist.edu.in

***Abstract*— Cloud computing provides the agility required to dynamically adjust applications in response to changing demands. Despite the proliferation of cloud service providers, the issue of vendor lock-in continues to pose significant challenges. Recent service disruptions have underscored the perils of exclusive reliance on a single provider. Existing cloud orchestration tools, while designed to mitigate these issues by facilitating deployments across multiple cloud providers, are often bound to provider-specific models. This necessitates users' familiarity with the unique offerings of each provider and may limit adaptability in the face of errors. To address these challenges, a custom wrapper is proposed that operates on top of Terraform, a leading Infrastructure-as-Code (IaC) tool. This wrapper enables diverse cloud deployments through a customized configuration file, simplifying the process of auditing, configuring, and securing various deployments. Practical experiments confirm the seamless deployment of infrastructure across various cloud providers, notably AWS and Azure. Specifically, the successful deployment of a Linux VM on both platforms showcases the adaptability of the approach to diverse cloud environments. The solution also maintains an easy learning curve, ensuring accessibility and industry-friendliness for users managing infrastructure across different cloud providers.**

Keywords*— *Terraform, Infrastructure-as-Code (IaC), Cloud orchestration

I. INTRODUCTION

Organizations are increasingly adopting a multi-cloud strategy, deploying their infrastructure across multiple cloud providers, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud [1-2]. This approach offers several benefits. Firstly, it enhances redundancy and reliability as data and applications can be distributed across different cloud platforms, reducing the risk of downtime or data loss. Secondly, it provides flexibility, enabling organizations to select the most suitable cloud services for specific workloads and applications. It also promotes cost optimization [3] by allowing companies to leverage competitive pricing and take advantage of cloud provider-specific strengths. Moreover, a multi-cloud strategy mitigates vendor lock-in [4], ensuring that an organization is not solely dependent on a single cloud provider. Overall, this approach empowers organizations with greater resilience, agility, and cost control in their cloud infrastructure operations.

A multi-cloud strategy, while offering advantages such as redundancy and flexibility, comes with significant disadvantages [5]. It can introduce complexity, increase costs due to management overhead and data transfer fees, pose challenges in maintaining security and compliance standards, require extensive vendor management efforts, and demand a diverse skill set. Interoperability issues, potential latency, and the risk of partial vendor lock-in are also concerns. Balancing these pros and cons, organizations must carefully plan and manage their multi-cloud deployments to ensure that the benefits outweigh the challenges and align with their specific objectives and resources.

To tackle some of the challenges, this research proposes creating a wrapper on top of Terraform, Infrastructure-as-code (IaC) [6] tool to deploy infrastructure across various cloud providers from a single configuration file that simplifies the management of multi-cloud environments. This approach abstracts the intricacies of individual cloud provider APIs, allowing users to define their infrastructure needs in one unified configuration. It promotes consistency, reducing the risk of configuration errors, and facilitates easier enforcement of security standards and compliance requirements.

Following this introduction, subsections explaining about the fields and technologies worked with in this study, has been summarized. Section II has the related works done in the field upon which knowledge and guidance are taken from for this research. Section III briefs the proposed solution. Section IV elaborates the model of the solution. Section V contains the data from experimentation with Terraform and Cloud Providers. Section VI summarizes the results and Section VII concludes the study. Section VIII gives the future scope of the research work.

A. Cloud Orchestration

Cloud orchestration is the management and coordination of various automated tasks and workflows within a cloud environment to ensure the efficient deployment, scaling, and management of complex applications and services [7]. It involves organizing and automating the interactions between multiple cloud resources, such as virtual machines, containers, storage, and networking components, to achieve a unified and orchestrated workflow. This process often utilizes orchestration tools and frameworks that help streamline the configuration, deployment, and scaling of

resources while maintaining consistency and ensuring optimal performance. Cloud orchestration plays a pivotal role in enabling scalability, resource optimization, and the seamless operation of cloud-based applications across diverse cloud environments..

B. *Infrastructure as Code*

IaC is a methodology that allows the management and provisioning of infrastructure resources using code and automation rather than manual processes. It involves defining and managing infrastructure—such as virtual machines, networks, storage, and configuration settings—through descriptive, human-readable files that can be version-controlled, shared, and executed by automation tools.

With IaC, instead of manually configuring or setting up infrastructure, developers and operations teams use code (often in declarative languages like YAML or JSON) to define the desired state of their infrastructure. Tools like Terraform, Ansible, and Chef interpret these code files to automatically create, modify, or delete resources across various cloud providers or on-premises environments.

This approach offers several benefits, including consistency, repeatability, and scalability. It ensures that the infrastructure's state is reproducible, making it easier to track changes, maintain version control, and replicate environments accurately. IaC also promotes collaboration between teams, enhances agility by simplifying and accelerating deployments, and reduces the likelihood of human error that can occur with manual configurations. Overall, Infrastructure as Code streamlines the management of complex infrastructures, making it a key practice in modern software development and system administration.

C. *TerraForm*

Terraform [8-9] is an open-source infrastructure-as-code (IaC) tool developed by HashiCorp. It is designed to automate the provisioning, configuration, and management of infrastructure resources, such as virtual machines, networks, storage, and other components, across various cloud providers (like AWS, Azure, Google Cloud), on-premises data centers, and other service providers. Terraform uses a declarative configuration language to define infrastructure as code, allowing users to specify the desired state of their infrastructure, and then Terraform automatically orchestrates the deployment and management of those resources to ensure they match the defined configuration.

IaC operates on crucial principles like idempotence, immutability, disposable systems, generic modules, and an ever-evolving design. Terraform seamlessly embraces these IaC principles by ensuring that changes are made only when necessary, promoting adaptability to hardware failures through dynamic resource provisioning. Its modular structure and generic modules enable a consistent and repeatable infrastructure approach [10]. Terraform's support for easy modifications in infrastructure configurations aligns well with the ever-evolving design principle, ensuring flexibility in meeting changing organizational needs.

Terraform offers crucial security features and best practices for securely managing infrastructure across various cloud environments. It enables users to integrate encryption, access controls, and network segmentation directly into their infrastructure code, ensuring consistent security measures. The integration with Identity and Access Management (IAM) services allows fine-grained control over resource permissions, enforcing least privilege principles to mitigate unauthorized access risks. Terraform Cloud ensures secure communication through end-to-end encryption with TLS [11] and maintains the confidentiality of sensitive information using encryption measures at rest. Authentication and authorization policies are strictly enforced for UI and API actions, contributing to a controlled and secure workflow. Terraform operations, such as planning and applying changes, occur in ephemeral environments, ensuring isolation and minimizing security risks. The reliability and availability of Terraform Cloud are further enhanced by its spread across multiple availability zones and regular data backups, ensuring the continuity of operations. In summary, Terraform's security features provide a robust foundation for managing infrastructure securely in diverse cloud environments.

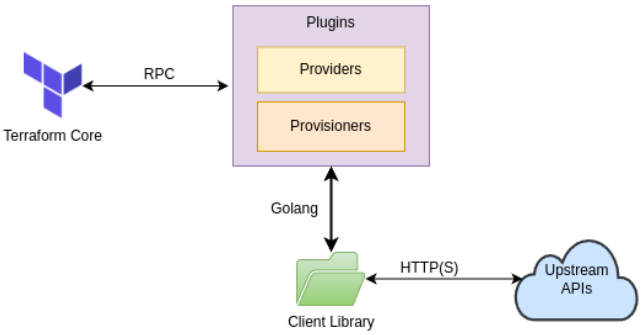

Fig.1 Terraform Core and Terraform Plugins

Figure 1 highlights how terraform manages dependencies and interactions between resources using its Resource Graph and communication between Terraform Core and Plugins. The Resource Graph outlines the order in which resources should be provisioned or updated to address dependencies. Terraform Core communicates with Provider Plugins [12], which handle authentication and define resources specific to infrastructure providers. This abstraction allows Terraform to manage dependencies between various cloud resources seamlessly. Additionally, Provisioner Plugins execute post-creation or destruction commands, facilitating coordinated interactions between resources across multiple cloud platforms. Overall, Terraform ensures a well-defined order of operations and consistent interactions in a multi-cloud environment through its modular architecture.

II. Related Works

In the expansive field of cloud orchestration, numerous studies and initiatives have contributed to advancing the management and coordination of cloud resources. Research and related work in this domain have focused on various aspects aimed at enhancing the efficiency, scalability, and resilience of cloud-based systems.

One area of related works involves the development of orchestration frameworks [13] and tools. These tools are designed to streamline the deployment and management of cloud resources across diverse platforms. Solutions such as Kubernetes, Docker Swarm, and Apache Mesos have

emerged as robust orchestration platforms, enabling the efficient deployment and scaling of containerized applications across cloud environments. [14-15] Moreover, there's ongoing research exploring how to extend these orchestration tools to manage hybrid and multi-cloud infrastructures, allowing seamless operation across multiple cloud providers.

Another significant focus of related work in cloud orchestration pertains to automation and self-healing capabilities. Researchers have been exploring ways to automate the orchestration of cloud resources, enabling systems to autonomously adapt to varying workloads and faults. Self-healing mechanisms [16-17] aim to detect and mitigate system failures automatically, ensuring continuous operations without human intervention. These approaches increase the reliability and availability of cloud-based services, essential in complex and dynamic cloud environments.

Moreover, the concept of federated and multi-cloud orchestration has been an area of extensive research. Federated cloud systems aim to interlink multiple clouds, enabling interoperability and resource sharing across diverse cloud providers. This research aims to address issues associated with data migration, security, and consistency in managing workloads and resources in federated environments. This line of work emphasizes ensuring seamless communication and coordination between different clouds to create a unified and efficient infrastructure.

Security and compliance in cloud orchestration have also been a key focus. Researchers have been exploring methods to ensure secure orchestration, implement effective governance, and maintain compliance across multiple cloud platforms [18]. This includes the development of security models, access control mechanisms, encryption, and auditing processes to safeguard data and applications while ensuring adherence to various regulatory standards in a multi-cloud setting.

Several solutions currently exist for orchestrating activities in the cloud. Containerization is one such method, but it requires an existing operating system and fully provisioned infrastructure. Notably, tools like Chef, Puppet, and Ansible function as Configuration Management tools, installing and managing software on pre-existing machines where resources are already provisioned. These tools employ agents on managed nodes, controllable from a centralized node using the OS that's already installed and provisioned. CloudFormation and Heat come close to infrastructure deployment by creating and provisioning resources in a cloud environment, but they are tied to specific providers like AWS or OpenStack [19].

III. Proposed Solution

The proposed solution streamlines infrastructure management by utilizing a dry Infrastructure as Code (IaC) file alongside a variable file containing customized parameters. A script will dynamically generate the variable file, capturing user inputs and storing them as variables. By leveraging both the variable and dry configuration files, a Terraform configuration language can be generated which can be then be utilized by Terraform to assess the current state of the cloud environment and seamlessly synchronize it according to the specified Terraform settings.

The proposed solution is using Terraform as a wrapper module. Unlike other tools, Terraform is provider-agnostic, meaning it works across various cloud providers. It uses Infrastructure as Code (IaC) to specify the desired infrastructure state, allowing for automatic deployment and management of resources. This includes managing both low-level components like compute, storage, and networking, and high-level components like DNS entries and Software as a Service (SaaS) features. This approach solves the limitations of previous tools and offers a flexible, provider-agnostic solution for infrastructure deployment and management.

IV. Proposed Model

A. System Architecture

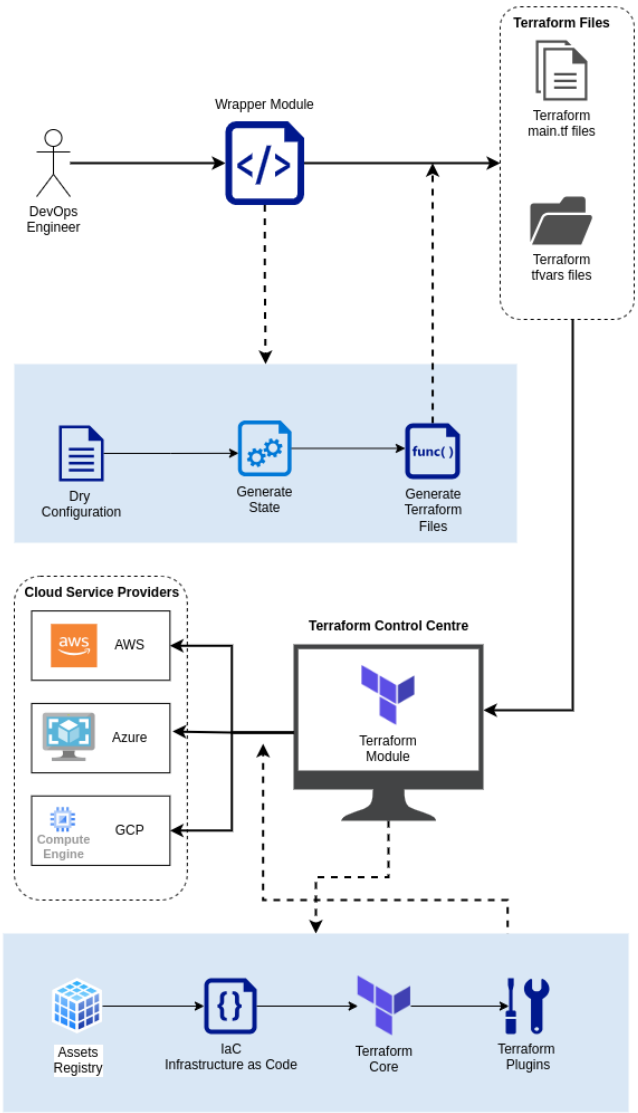

Fig. 2 Architecture Diagram

To enable the transition from the dry code to dynamic Terraform code, a 'Wrapper Module' is introduced. This module leverages Terraform variables, Terragrunt [20] and bash scripting to enable dynamic resource deployment. Figure 2 highlights the system architecture that helps to integrate the Wrapper Module for orchestrating dynamic resource deployment.

A. Creating a Modular Dry Code

To achieve modularity, a 'dry code' template is established. This template abstracts provider-specific details, offering a standardized foundation that can be easily customized for different cloud environments.

B. Utilizing Terraform Variables

Terraform variables play a crucial role in the methodology. They serve as placeholders for dynamic information that can be adjusted for each cloud provider. These variables are used for configuring attributes like instance types, regions, and API keys.

C. Implementing Terragrunt and Bash Scripting

Terragrunt, coupled with bash scripting, plays a pivotal role in transforming the dry code into dynamic Terraform code. Terragrunt serves as the engine responsible for constructing the Terraform configuration. Bash scripting manages dynamic inputs and facilitates the selection of Terraform variables based on provider-specific requirements.

V. EXPERIMENTS

Terraform is heavily reliant on the concept of "state" in order to effectively manage and track resources within a cloud environment. Acting as a "state-level worker," Terraform takes charge of the infrastructure state by continuously comparing the current state of deployed resources with the specified state outlined in the configuration files. It then takes necessary measures to ensure that both states are aligned accordingly. This robust state management is crucial to Terraform's overall functionality, as it guarantees precise monitoring of resource details, configurations, and interrelationships. In its role as a "state-level worker," Terraform plays a pivotal part in infrastructure as code workflows, consistently maintaining the synchronization between the actual infrastructure and the designated configuration.

TABLE 1 EXPERIMENT PARAMETERS

	AWS	**Azure**
Module	hashicorp/aws-5.32.1	hashicorp/azurerm-3.77.0
Region	ap-south-1	westindia
VPC	10.0.0.0/16	10.0.0.0/16
Subnet	10.0.1.0/24	10.0.2.0/24
Port Forwarding	22,80	22
Instance Type	t2.micro	Standard_82ms
OS	Amazon Linux 2023	Ubuntu 20.04 LTS

In accordance with the infrastructure specifications outlined in Table 1, a tailored variable file was formed to use it with the dry configuration file. During the execution of the orchestration engine, details regarding the machine state were gathered. The experiment utilized a system featuring an AMD Ryzen 5 4600H processor, 16GB of RAM, and running Ubuntu 20.04 LTS as the primary operating system. To gather the usage report, tools such as bpytop, htop, lsblk, iotop, iftop and free [21] were sequentially utilized. The input values from the table were obtained using a shell script to create the variable file, which was then combined with Terragrunt to create variables specific to Terraform configuration language. In the end, the infrastructure of the cloud environment was deployed using the final variable file by Terraform.

VI. RESULTS

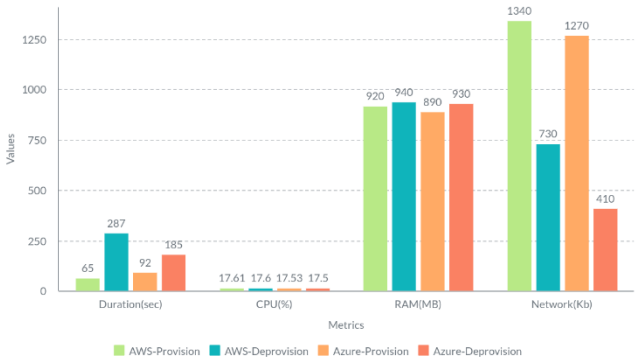

Fig. 3 Consolidated Results of Executions

The experiment highlights the effective utilization of Terraform and Terragrunt working together in addressing the specified problem. Figure 3 provides a comprehensive overview of system usage reports for both the AWS and Azure modules of Terraform during the provisioning and deprovisioning of cloud environments. The data presented in the figure represents averages derived from multiple execution to establish a baseline. Notably, the experiment indicated that Azure consumed a comparatively longer duration for both provisioning and deprovisioning processes. Despite variations in runtime, CPU, and RAM utilization remained nearly same across all types of provisioning and deprovisioning activities. Network utilization peaked at 1340kb/s during provisioning and reached a low of 410kb/s during deprovisioning. These findings offer insights into the performance characteristics and comparative efficiencies of Terraform for the specified experimental parameters.

VII. CONCLUSION

The proposed solution, an extension built upon the widely adopted Terraform cloud orchestration tool, is particularly well-suited for real-time implementation. Terraform's dominant market share of over 30.50%, backed by a substantial user base of more than 35,210 customers, signifies its popularity and trust within the industry. With an impressive 40.2k stars and contributions from over 1,785 developers on GitHub, Terraform boasts robust community support and continuous improvement. These statistics underscore its suitability for real-time scenarios. Moreover, the tool's exceptional download numbers, with 15.6 million downloads in one week and 657.3 million downloads in a year, highlight its widespread usage and reliability. Leveraging Terraform's proven success, this solution introduces a simple wrapper module with Terragrunt and manual scripts, demonstrating its potential to significantly streamline provisioning and de-provisioning operations in real-time implementations. In testing phase, the solution

effectively deployed simple infrastructure across multiple cloud platforms, demonstrating its capability to address multi-cloud deployment challenges.

VIII. FUTURE WORK

Future work needs to be done on the field of Cloud Orchestration and an open standard can be developed in the industry to tackle the problem of cloud providers inter-operable nature to increase ease in auditing and maintenance of the cloud deployments.

The biggest challenges in Multi Cloud Orchestration are offerings by the Cloud providers being changed frequently which leads to complexities in managing the infrastructure through a unified interface.

The interoperability and the specification will play a vital role in developing such solutions. Cloud Providers should push towards standardization of the common offerings and naming them to reduce confusion in migration from one provider to another.

REFERENCES

[1] AWS Docs. AWS. https://docs.aws.amazon.com/, 2023. Accessed on 2023-12-09.

[2] Azure Docs. Azure. https://learn.microsoft.com/en-us/azure/, 2023. Accessed on 2023-12-09.

[3] D. Chemodanov, P. Calyam, S. Valluripally, H. Trinh, J. Patman and K. Palaniappan, "On QoE-Oriented Cloud Service Orchestration for Application Providers," in IEEE Transactions on Services Computing, vol. 14, no. 4, pp. 1194-1208, 1 July-Aug. 2021, doi: 10.1109/TSC.2018.2866851.

[4] D. Baur, D. Seybold, F. Griesinger, H. Masata and J. Domaschka, "A Provider-Agnostic Approach to Multi-cloud Orchestration Using a Constraint Language," 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Washington, DC, USA, 2018, pp. 173-182, doi: 10.1109/CCGRID.2018.00032.

[5] D. Sitaram et al., "Orchestration Based Hybrid or Multi Clouds and Interoperability Standardization," 2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), Bangalore, India, 2018, pp. 67-71, doi: 10.1109/CCEM.2018.00018.

[6] J. C. Patni, S. Banerjee and D. Tiwari, "Infrastructure as a Code (IaC) to Software Defined Infrastructure using Azure Resource Manager (ARM)," 2020 International Conference on Computational Performance Evaluation (ComPE), Shillong, India, 2020, pp. 575-578, doi: 10.1109/ComPE49325.2020.9200030.

[7] Linh Manh Pham, Alain Tchana, Didier Donsez, Noel de Palma, Vincent Zurczak, et al.. Roboconf: a Hybrid Cloud Orchestrator to Deploy Complex Applications. 2015 IEEE 8th International Conference on Cloud Computing, Jun 2015, New York, United States. ff10.1109/CLOUD.2015.56ff. ffhal-01228353.

[8] Terraform Docs. Hashicorp. https://developer.hashicorp.com/terraform/docs, 2023. Accessed on 2023-12-09.

[9] Y. Brikman. Terraform: Up and Running: Writing Infrastructure as Code. O'Reilly Media, 2017.

[10] L. R. de Carvalho and A. Patricia Favacho de Araujo, "Performance Comparison of Terraform and Cloudify as Multicloud Orchestrators," 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, VIC, Australia, 2020, pp. 380-389, doi: 10.1109/CCGrid49817.2020.00-55.

[11] Security Model - Terraform Cloud | Terraform | HashiCorp Developer. Hashicorp. https://developer.hashicorp.com/terraform/cloud-docs/architectural-details/security-model, 2023. Accessed on 2023-12-09.

[12] Plugin Development - How Terraform Works With Plugins | Terraform | HashiCorp Developer. Hashicorp. https://developer.hashicorp.com/terraform/plugin/how-terraform-works, 2023. Accessed on 2023-12-09.

[13] Tomarchio, O., Calcaterra, D. \& Modica, G.D. Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks. J Cloud Comp 9, 49 (2020). https://doi.org/10.1186/s13677-020-00194-7

[14] S. Jarrous-Holtrup, S. Gorlatch, M. Dey and F. Schamel, "Multi-Cloud Container Orchestration for High-Performance Real-Time Online Applications," 2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Naples, Italy, 2023, pp. 307-313, doi: 10.1109/PDP59025.2023.00054

[15] Soltesz S, Pötzl H, Fiuczynski ME, Bavier A, Peterson L (2007) Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors In: Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007. EuroSys '07, 275–287.. ACM, New York. https://doi.org/10.1145/1272996.1273025.

[16] A. Mayoral et al., "Control orchestration protocol: Unified transport API for distributed cloud and network orchestration," in Journal of Optical Communications and Networking, vol. 9, no. 2, pp. A216-A222, Feb. 2017, doi: 10.1364/JOCN.9.00A216.

[17] Dai, Yiwu & Xiang, Yanping & Zhang, Gewei. (2009). Self-healing and Hybrid Diagnosis in Cloud Computing. 5931. 45-56. 10.1007/978-3-642-10665-1_5.

[18] Nicolae Paladi, Antonis Michalas, and Hai-Van Dang. 2018. Towards Secure Cloud Orchestration for Multi-Cloud Deployments. In Proceedings of the 5th Workshop on CrossCloud Infrastructures \& Platforms (CrossCloud'18). Association for Computing Machinery, New York, NY, USA, Article 4, 1–6. https://doi.org/10.1145/3195870.3195874

[19] Terraform vs. Alternatives | Terraform | HashiCorp Developer. Hashicorp. https://developer.hashicorp.com/terraform/intro/vs, 2023. Accessed on 2023-12-09.

[20] Terragrunt Docs. Terragrunt. https://terragrunt.gruntwork.io/docs/, 2023. Accessed on 2023-12-09.

[21] Linux man pages. Die.net. https://linux.die.net/man/, 2023. Accessed on 2023-12-09.