Original Article
ISSN (Online): 2582-7472

ShodhKosh: Journal of Visual and Performing Arts
March 2024 5(3), 1385–1393

IAC FRAMEWORK DEVELOPMENT FOR EC2 MIGRATION

Prevesh Kumar Bishnoi [1], Dr. Dharmender Kumar [2], Dr. Prateek Bhanti [3]

[1] Computer Science and Engineering, School of Engineering and Technology Mody University of Science and Technology
[2] GJUST Hisar
[3] Computer Science and Engineering, School of Engineering and Technology Mody University of Science and Technology

DOI
10.29121/shodhkosh.v5.i3.2024.4541

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

ABSTRACT

The evolution of a Terraform imported Infrastructure as Code (IAC) framework for EC2 migration investigated in this research. It guarantees consistency, automates deployment, and boosts scalability, hence lowering hand-made errors. Comparative study of the framework with conventional, non-IAC methods shows effectiveness increases. It is found that there are various advantages are error free, faster provisioning, than manual migration. Manual provisioning without IAC generates higher risk, more running costs, and discrepancies. The analogy underlines IAC's changing capacity in cloud computing management. AWS, Azure and GCP and local Virtual box are selected for migration and inter cloud migration.

Keywords: Migration, AWS, Azure, GCP, IAC, Terraform, Inter-Cloud

1. INTRODUCTION

The increasing use of cloud computing has transformed IT infrastructure management and made companies able to effectively expand under control of expenses. Among the most used cloud services, Amazon Elastic Compute Cloud (EC2) offers under programmable settings on-demand processing capacity. Still, hand operating EC2 instances can lead to major issues including operational inefficiency, configuration drift, and human mistakes. As companies expand their cloud environments and drive unequal infrastructure rollout and additional maintenance costs, these issues become increasingly more apparent.

By means of declarative language, infrastructure as language (IAC) has evolved into a potent tool for controlling cloud architecture enabling companies to specify and automate provisioning strategies. Version-activated infrastructure definitions let IAC guarantee consistency, lowers deployment time, and—by allowing it—helps to promote teamwork. Thanks to its cloud-agnostics, modularity, and declarative syntax, Terraform has become rather popular among many IAC tools. Terraform dramatically lowers the need for regular manual interventions by codifying instance configurations, network setups, and security rules into reusable scripts, therefore enabling EC2 migration.

This paper aims to evaluate, utilising Terraform into an IAC framework, the advantages of moving EC2 instances against conventional, non-IAC techniques. By allowing businesses to apply scalable, automated, error-free deployments using Terraform, infrastructure dependability and security will be much improved. The paper also looks at how Terraforms infrastructure architecture, state management, and resource dependencies simplify migration processes thereby guaranteeing little downtime and faultless transitions.

On the other hand, conventional manual deployment systems inspire managers to independently construct EC2 instances via the AWS Management Console or CLI. This method not only takes time but also creates operational inefficiencies and security issues since every instance may have little differences in settings, therefore creating inconsistencies. Furthermore, challenging without IAC is correct undo of installations or duplicate configurations. Observing changes in infrastructure becomes challenging. In this study, Terraform-based EC2 migration against hand deployment is evaluated in terms of efficiency, scalability, cost, and risk decreasing.

The results of this research highlight the need of using IAC models for companies trying to maximise cloud operations and guarantee efficient infrastructure management.

Companies can assure compliance, increase agility, and encourage interaction among development teams and operations by means of an IAC design. Since Terraform lets code-driven automation create, modify, and govern cloud resources, it is the perfect tool for operations involving EC2 migration. Since it shows how IAC adoption can affect the maintenance and provision of cloud infrastructure, the comparison study emphasises even more the need of IAC adoption.

Lastly, this paper offers understanding of how Terraform-based IAC systems streamline EC2 migrations, so guaranteeing a perfect, scalable, and efficient cloud environment while reducing risks related with hand-based infrastructure management.

2. REVIEW

The way Terraform emerged via multi-cloud deployment has revolutionised infrastructure management. Although some companies find the change to Terraform difficult, its automated features greatly reduce related error risk, manual participation, and deployment time. Terraform makes infrastructure deployment across numerous environments easier by supporting several cloud providers including AWS, Azure, and Digital Ocean [1]. Disaster recovery is one place Terraform most certainly shows benefit. While traditional disaster recovery techniques are sometimes slow and expensive, Terraform paired with Kubernetes maximises recovery by cutting infrastructure maintenance time and processes. While Kubernetes further reduces deployment time by roughly 530%, Adopting Terraform cuts the recovery process time by 235%, therefore producing a total reduction in disaster recovery length compared to manual solutions [2].

In development, staging, and production settings, Terraform also helps to preserve infrastructure consistency. Automating versioning and resource procurement helps companies to assure regulatory compliance, prevent configuration drift, and monitor changes. Moreover, by connecting with AWS CloudFormation and AWS Service Catalogue, Terraform offers further degrees of control for cloud resource management [3]. As companies increase their cloud operations, IaC tool scalability raises increasingly challenging issues. Comparative assessments of Terraform and other IaC technologies show that it provides better scalability, hence it is the perfect solution for companies needing major infrastructure automation. Visualisation tools allow companies to evaluate scalability criteria, therefore influencing their choice of the most appropriate solution for their requirements [4].

Another problem Terraform tackles is business continuity to boost acceptance of clouds. Running and maintaining physical backup locations can be costly even if Terraform-based cloud installations offer reasonably priced, automated disaster recovery options. Notwithstanding these advantages, cloud adoption presents difficulties in several domains, like Africa, where institutions fight adoption owing to infrastructural limitations.

Inspired by IaC, a system suggested for Kenya aims to lower human error, increase efficiency, and minimise maintenance expenses [5]. 75% of the industry is predicted to be multi-cloud based, so as cloud computing develops multi-cloud solutions are becoming standard. For such settings, Terraform is ideal; it outperforms rivals like Cloudify in pragmatic tests.

First selected by firms using multi-cloud architectures is its capacity to handle difficult cloud orchestrating responsibilities [6].

Edge computing reduces latencies by bringing cloud services closer to clients. By cross-ISA migration of natively generated containers, H-Container allows service migration to the closest edge node, constrained but necessary by various Instruction Set Architectures (ISAs). Working in user space, Linux-compliant, it needs no source-code change. Combining LLVM, CRIU, and Docker, H-Container adds just 10–100 ms during migration and increases Redis throughput by 94%. [10] Function-as- a- Service (FaaS) streamlines cloud operations—albeit only for stateless activities. Cloudburst connects low-latency mutable state FaaS to stateful apps. It mixes function executors for maximum data locality using Anna, an autoscaling key-value store. By means of lattice-encapped state and distributed session protocols, Cloudburst guarantees cache consistency. It promotes serverless computing for a stateful activity and lowers state-management overheads. [12]

3. PROPOSED DESIGN

3.1. LOCAL VM TO AWS, AZURE, GCP

Migrating a VirtualBox VM to cloud platforms such as AWS, Azure, or GCP requires a structured process to ensure compatibility and smooth deployment. The user initiates the migration by requesting VirtualBox to export the VM in a compatible format such as OVA, VHD, VMDK, or RAW, depending on the cloud provider's requirements. After exporting, the VM image is sent to a conversion tool that processes and converts it into a format suitable for cloud deployment. This level assures that the picture fits the specific infrastructure of the given cloud platform. Either Azure Migrate or GCP Cloud Import let to publish the VM image to the destination cloud platform either following conversion, import and export tools for AWS VM Import/Export. The cloud platform makes sure the selected virtual machine image meets particular criteria before being displayed as a live instance. After successful provisioning, the cloud platform alerts the user; she or him then looks at the virtual machine to confirm the successful migration. Azure follows different GCP criteria and utilizes VMDK; AWS requests VHD, or RAW. Perhaps upgrades in BIOS/UEFI, storage efficiency, and network settings would guarantee proper compatibility during transfer. Three automated solutions for Compute Engine that help to properly manage conversions, uploads, and provisioning so simplifying the process are AWS Server Migration Service, Azure Migrate, and GCP Migrate. Usually, network conditions, CPU architecture, disc format define seamless transfer. Some VMs may need further configuration compatible with the guidelines for the cloud architecture before deployment. Maintaining system integrity and efficiency, the sequence diagram reveals a methodical and successful virtual machine transfer from VirtualBox to the cloud platform. Fig. 3.1 is representing it.

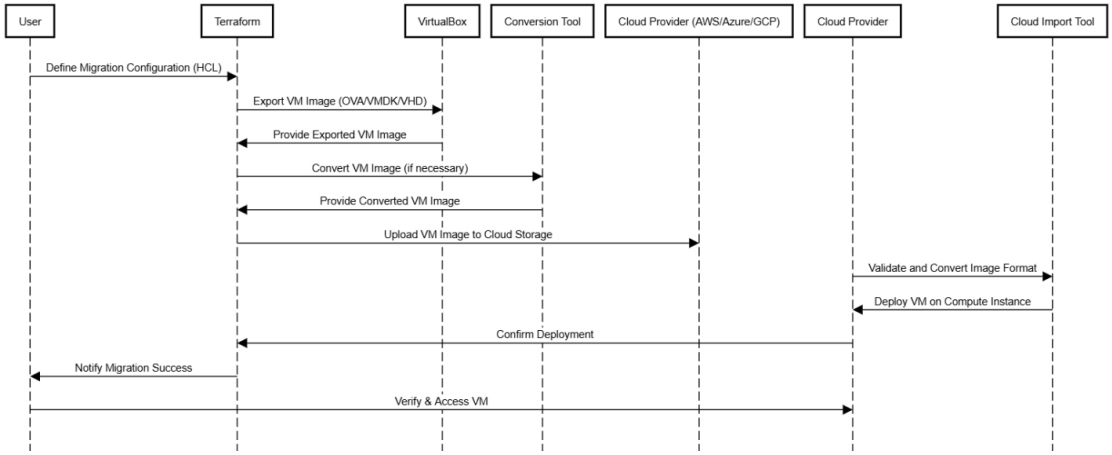

Fig 3.1 Local VM to AWS, Azure or GCP

3.2. AWS, AZURE, GCP TO VM LOCAL

Moving a virtual machine from cloud platforms such as AWS, Azure, or GCP to a local environment follows a rigorous method assured to be compatible and simple deployment. Sequence diagram of migration is given in Fig 3.2. The process starts when the user requests a virtual machine export from the cloud platform, generating the VM image either VHD, VMDK, or RAW. Once the image becomes available, the cloud platform allows the user download it therefore allowing access to the exported virtual machine. Should it be necessary, the image is treated with a local conversion tool to ensure fit with the target hypervisor—VirtualBox or VMware. The converted VM then imports into the local VM environment

where, where necessary, network, storage, and CPU parameters are applied. Following efficient import, the local virtual machine environment signals the user to show the migration is complete. Finally comes the user verifying and operating on the virtual machine to ensure proper conditions. Preserving system integrity and performance this way ensures that the migrated virtual machines run as expected in the local environment.

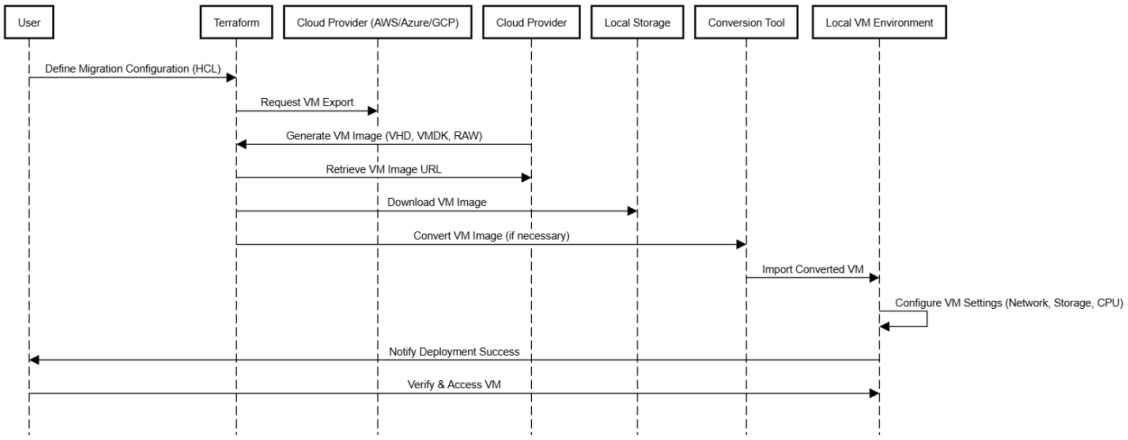

Fig 3.2 AWS, Azure or GCP to Local VM

3.3. AWS, GCP AND AZURE INTERMIGRATION

Moving a virtual machine to AWS, Azure, and GCP requires a well defined series of procedures guaranteed fit and perfect deployment. Different VM image forms and migration tools are used on every cloud platform; hence, export, conversion, and import processes have to be properly handled. Typically starting with a virtual machine export from the source cloud provider, the user wants to extract the VM image in a supported form including VHD, VMDK, or RAW. Once exported, the user downloads the virtual machine image and is ready to upload it to the selected cloud provider. Depending on the migration technique, additional conversions could be required to meet the standards of the destination platform. Sequence diagram of migration is given in Fig 3.3.1, and Fig 3.3.2 showing the migration AWS to Azure and AWS to GCP. Easy deployment of the virtual machine by reviewing and improving it as well as automated solutions including AWS VM Import / Export, Azure Migrate, GCP Cloud Import / Export help to facilitate this change.

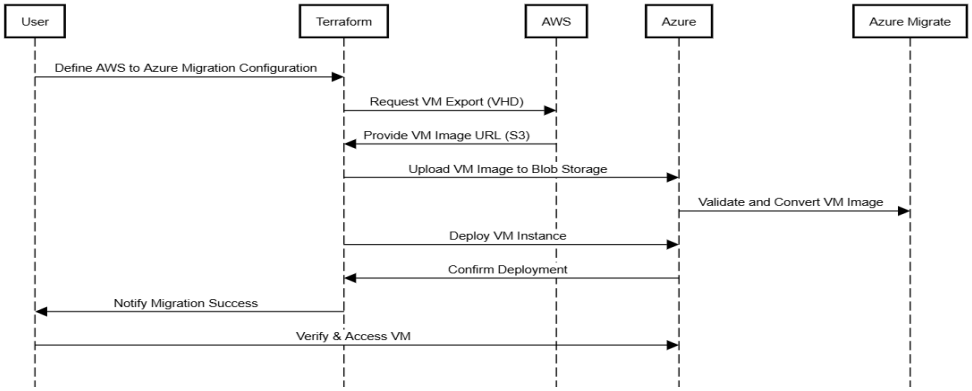

Fig 3.3.1 AWS to Azure Intermigration

s

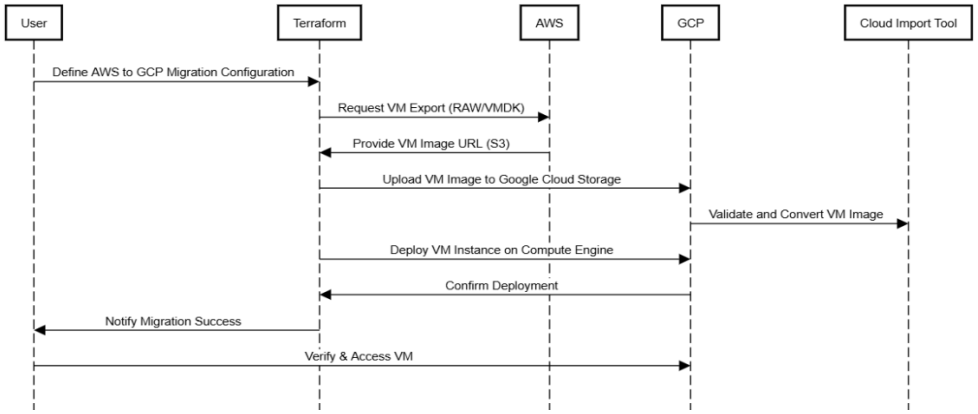

Fig 3.3.2 AWS to GCP Intermigration

After image preparation, it ends up in Amazon S3, Azure Blob Storage, Google Cloud Storage, or another such, the storage system of the target cloud provider. The cloud platform turns the virtual machine into a fit for the new surroundings by processing the image using its own import tool. This phase is fairly critical since it ensures virtual machine integrity, network configurations, and storage mappings. Once the translation is completed, the virtual machine is provisioned in the target cloud architecture where numerous elements including security limitations, network settings, and CPU optimisation can be applied. Throughout the transfer, cloud systems execute compatibility tests all around to avoid issues originating from several BIOS/UEFI boot settings, network topologies, and disc formats. Sequence diagram of migration is given in Fig 3.3.3, and Fig 3.3.4 showing the migration Azure to AWS and Azure to GCP to GCP.

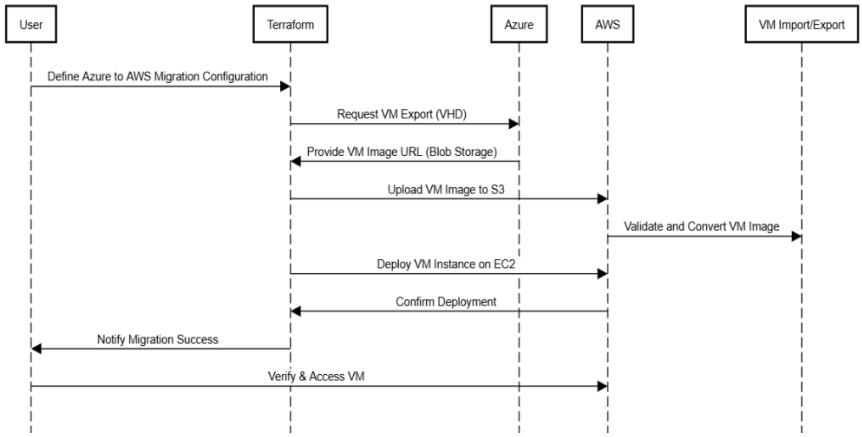

Fig 3.3.3 Azure to AWS Intermigration

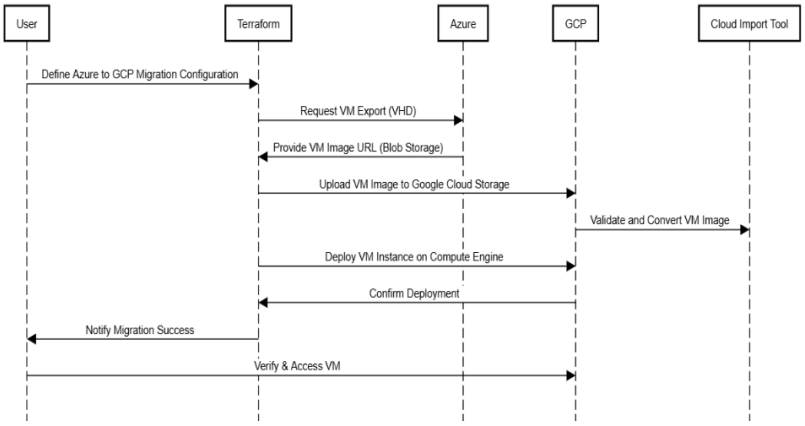

Fig 3.3.4 Azure to GCP Intermigration

The user is notified and last tests are carried out to guarantee correct operation after the virtual machine is practically located on the new cloud platform. Examining its performance, interface fit in the new surroundings, the user visits the moved virtual machine. Among the sometimes necessary post-migration changes are resizing events, changing drivers, and creating security groups. The methodical technique of the UML sequence diagrams provides lowest operational efficiency and downtime by means of exactly directing cloud-to- cloud migration. As multi-cloud use increases to satisfy scalability, flexibility, and cost optimisation while retaining system integrity spanning many cloud platforms, businesses rely on these migration solutions. Sequence diagram of migration is given in Fig 3.3.5, and Fig 3.3.6 showing the migration GCP to AWS and GCP to azure.

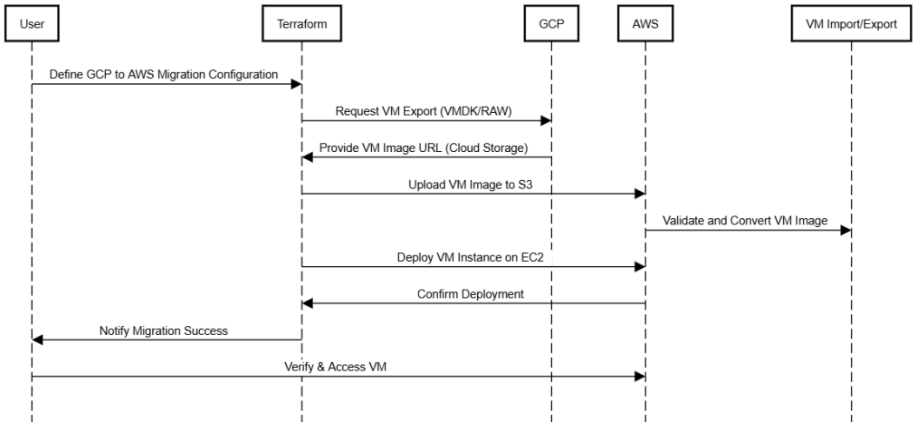

Fig 3.3.5 GCP to Azure Intermigration

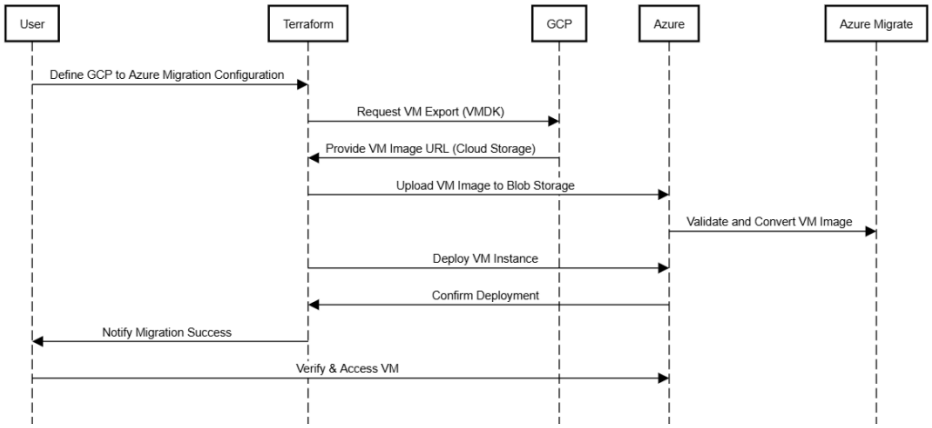

Fig 3.3.6 GCP to Azure Intermigration

3.4. FRAMEWORK MODULE DEVELOPED IN TERRAFORM AND TESTED

3.4.1. INTERCLOUD

Eliminating human involvement, the Terraform module for cloud virtual machine migration automates the entire virtual machine migration process across AWS, Azure, and GCP. Users of Terraform's Infrastructure as Code (IaC) can create a migration pipeline by which VMs are exported from the source cloud, uploaded to cloud storage (AWS S3, Azure Blob Storage, or GCP Cloud Storage), and later used in the destination cloud environment. Using Terraform's Null-Resource and Local-Exec capability, the module generates CLI commands contacting cloud provider APIs, hence guaranteeing effective virtual machine management. Apart from offering multi-cloud compatibility, the module lets users change the targeted cloud instance types, compute configurations, and storage accounts. AWS VM Import/Export, Azure Migrate, GCP Cloud Import assures to keep VM resources under control through cloud-specific rules. Automating virtual machine migration with Terraform helps companies simplify cloud adoption, improve disaster recovery planning, and reduce migration downtime. This module will particularly help businesses implementing a multi-cloud strategy as it allows fast, repeatable, scalable virtual machine migration between cloud providers. Unlike traditional migration solutions with fundamentally manual approaches and compatibility challenges, this Terraform module abstracts

complexity and offers a disciplined way of provisioning and infrastructure management. Since its outputs fully reflect the state of the migration, the module guarantees traceability and openness. Fig 3.5.1 and Fig 3.5.2 representing the structure of the modules developed terraform.

```
Cloud_vm_migration/
├── modules/
│   ├── aws_to_azure/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── outputs.tf
│   │   └── README.md
│   │
│   ├── aws_to_gcp/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── outputs.tf
│   │   └── README.md
│   │
│   ├── azure_to_aws/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── outputs.tf
│   │   └── README.md
│   │
│   ├── azure_to_gcp/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── outputs.tf
│   │   └── README.md
│   │
│   └── gcp_to_azure/
│       ├── main.tf
│       ├── variables.tf
│       ├── outputs.tf
│       └── README.md
│
├── examples/
│   ├── aws_to_azure.tf
│   ├── aws_to_gcp.tf
│   ├── azure_to_aws.tf
│   ├── azure_to_gcp.tf
│   ├── gcp_to_aws.tf
│   └── gcp_to_azure.tf
│
├── scripts/
│   ├── export_vm.sh
│   ├── convert_vm.sh
│   ├── upload_vm.sh
│   └── deploy_vm.sh
│
├── main.tf
├── variables.tf
├── outputs.tf
└── README.md
```

Fig 3.5.1 Terraform Framework Structure Cloud Provider to VM

4. RESULTS

4.1. RESULTS OF TESTING FOR MODULES

From local to cloud and cloud to local—the graph and table show a comparison between Terraform and manual VM migration across several cloud providers—including AWS, Azure, and GCP in both directions. The line graph clearly shows the large time variations; Terraform often lowers migration length as compared to hand-made methods. Terraform automates export, upload, conversion, and deployment, so simplifying the process even if hand migration calls for several manual processes, so extending execution times. According the research, Terraform completes most migrations in 15 to 33 minutes; manual processes for a 10 GB virtual machine image take 45 to 79 minutes. Larger virtual machine files clearly show the differences since human mistakes, network restrictions, and extra reconfiguration efforts take up to five times longer manually. Fig. 4.1.1 represent the comparison of manual and terraform module based migration. Data for the same is given in Table 4.1.1.

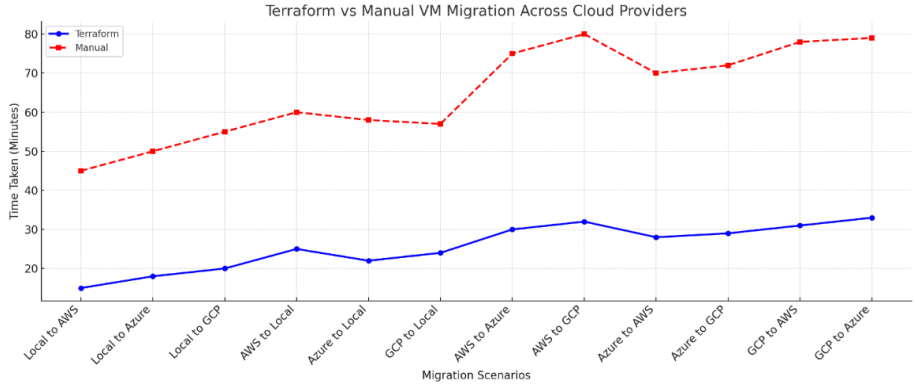

Fig. 4.1.1 Graphical Comparison of Manual and Terraform based migration

Table: 4.1.1 Comparison of Migration Time

Migration Scenario	Terraform Time (min)	Manual Time (min)
Local to AWS	15	45
Local to Azure	18	50
Local to GCP	20	55
AWS to Local	25	60
Azure to Local	22	58
GCP to Local	24	57
AWS to Azure	30	75
AWS to GCP	32	80
Azure to AWS	28	70
Azure to GCP	29	72
GCP to AWS	31	78
GCP to Azure	33	79

Direct comparisons between Terraform and human processes are made feasible by extensive study of migration timeframes given by the table for numerous situations. It demonstrates how Terraform offers scalable, consistent migration timeframes; manual migration suffers more volatility and inefficiencies. A manual approach takes 45 minutes, so showing a 3x efficiency increase; Terraform takes 15 minutes for a Local-to---- AWS migration. Terraform saves up to 50% of the time in same manner with cross-cloud migration—AWS to GCP, Azure to AWS, etc.). Moreover, Terraform remains helpful when virtual machine capacity increases—that is, 50 GB or 100 GB—while hand operations start to become really slow. Since this comparison underlines its main advantages, terraform automation is the advised approach for multi-cloud strategies, disaster recovery, and large-scale cloud migration.

Estimated Migration Times for Different File Sizes

If we assume a linear scaling based on file size, approximate times for Terraform vs Manual Migration can be estimated in Table 4.1.2.:

Table: 4.1.2 Migration Time with Different File Size

File Size (GB)	File Size (MB)	Terraform Time (min)	Manual Time (min)
10 GB	10,000 MB	15 - 33 min	45 - 79 min
20 GB	20,000 MB	30 - 66 min	90 - 158 min
50 GB	50,000 MB	75 - 165 min	225 - 395 min
100 GB	100,000 MB	150 - 330 min	450 - 790 min

5. CONCLUSION

Terraform provides an efficient and automated solution for VM migration between AWS, Azure, GCP, and local environments. It significantly reduces migration time compared to manual methods, which are slow and error-prone. The graph and table confirm that Terraform completes migrations 2-5 times faster than manual approaches. As VM sizes increase, manual migration slows down, while Terraform maintains consistent performance. Using Terraform's Infrastructure as Code (IaC) approach ensures repeatability, scalability, and reduced operational overhead. This makes it ideal for multi-cloud strategies, disaster recovery, and enterprise cloud transitions. Standardizing Terraform across several cloud providers allows businesses to migrate activities with least of impact. By means of the full migration process, it minimizes human errors and optimizes costs. Terraform is the advised technology for cloud migration as the automation increases dependability and scalability.Terraform shines all throughout in speed, dependability, and efficiency over hand migration.

CONFLICT OF INTERESTS

None.

ACKNOWLEDGMENTS

Authors want to thanks to their organizations, GJUST Hisar and Mody University of Science and Technology to providing the best environment for research and development

REFERENCES

A. Mehdi; R. Walia,''Terraform: Streamlining Infrastructure Deployment and Management Through Infrastructure as Code'',IEEE Conferences,2023

R. B. Bahaweres; F. Muhammad Najib,''Provisioning of Disaster Recovery with Terraform and Kubernetes: A Case Study on Software Defect Prediction'',IEEE Conferences,2023

S. Sharma; P. Agarwal; R. Tyagi,''High Level Cloud Architecture for Automated Deployment System Using Terraform'',IEEE Conferences,2023

M. K. Bali; R. Walia,''Enhancing Efficiency Through Infrastructure Automation: An In-Depth Analysis of Infrastructure as Code (IaC) Tools'',IEEE Conferences,2023

A. Dalvi,''Cloud Infrastructure Self Service Delivery System using Infrastructure as Code'',IEEE Conferences,2022

N. PetroviÄ‡; M. Cankar; A. Luzar,''Automated Approach to IaC Code Inspection Using Python-Based DevSecOps Tool'',IEEE Conferences,2022

A. -F. Sicoe; R. Botez; I. -A. Ivanciu; V. Dobrota,''Fully Automated Testbed of Cisco Virtual Routers in Cloud Based Environments'',IEEE Conferences,2022

J. DesLauriers; J. Kovacs; T. Kiss,''Abstractions of Abstractions: Metadata to Infrastructure-as-Code'',IEEE Conferences,2022

J. Eickhoff; J. Donkervliet; A. Iosup,''Meterstick: Benchmarking Performance Variability in Cloud and Self-hosted Minecraft-like Games'',IEEE Conferences,2022

Xing2022,''H-Container: Enabling Heterogeneous-ISA Container Migration in Edge Computing'',ACM Trans. Comput. Syst.,2022

S. Muthoni; G. Okeyo; G. Chemwa,''Infrastructure as Code for Business Continuity in Institutions of Higher Learning'',IEEE Conferences,2021

Sreekanti2020,''Cloudburst: stateful functions-as-a-service'',Proc. VLDB Endow.,2020

L. R. de Carvalho; A. Patricia Favacho de Araujo,''Performance Comparison of Terraform and Cloudify as Multicloud Orchestrators'',IEEE Conferences,2020

J. C. Patni; S. Banerjee; D. Tiwari,''Infrastructure as a Code (IaC) to Software Defined Infrastructure using Azure Resource Manager (ARM)'',IEEE Conferences,2020

D. Vladusic; D. Radolovic,''Infrastructure as Code for Heterogeneous Computing'',IEEE Conferences,2020

J. M. Bradshaw,''Terraforming cyberspace'',IEEE Conferences,2001