International Journal of Engineering & Tech Development
ISSN: 3069-672X/ Open Access / IJETD_Y1F1A003
Apex Academia Press / https://doi.org/10.63665/ijetd-y1f1a003

Original Article

A Framework for Multi-Cloud Network Orchestration Using Ansible and Terraform in Hybrid Environments

Purna Chandra Rao Chinta[1], Chethan Sriharsha Moore[2]

[1 2]Microsoft, Support Escalation Engineer

Abstract

With everything going digital, companies rely nowadays on more hybrid and multi-cloud environments, which give them more strength, flexibility, and room to grow. Given the fact that each cloud platform has its own settings, APIs, and services, it is challenging to keep track of network infrastructure across cloud platforms. This paper proposes the use of Terraform for infrastructure provisioning and Ansible for configuration management in order to obtain an enhanced network management in multi-cloud hybrid environments. The framework ensures consistency in terminology to refer to network resources across all platforms, such as AWS, Azure, and on-premise data centres, while it also enables the possibility of automatic deployment. The proposed methodology uses Ansible along with Terraform to create testable, changeable, and reproducible deployments. This ensures smoother operations and eliminates the possibility of getting stuck with a vendor. For hybrid infrastructures, automatic configuration of VPN, firewall, and routing configurations is possible with the framework. Following is a real example.

Keywords

Multi-Cloud, Hybrid Cloud, Network Orchestration, Infrastructure as Code (IAC), Ansible, Terraform, Cloud Automation, DevOps, Cloud Provisioning.

Article History

Received: 02.02.2025

Accepted: 27.02.2025

Published: 15.03.2025

1. Introduction

A. Overview of Hybrid and Multi-Cloud Environments

Speed, flexibility, and compliance are some of the reasons why hybrid and multi-cloud environments are becoming increasingly utilized in business. A typical hybrid cloud model would include public cloud services integrated with private infrastructure hosted on-premise. An organization can operate from wherever it gets the job done efficiently, cost-effectively, and in a mission-critical manner. Multi-cloud just means using more than one public cloud service-included but not limited to AWS, Azure, and Google Cloud-so you do not have to deal with just one vendor. You will have the ability to leverage features available only from a single provider and achieve easier access to services. While these models afford businesses more freedom than ever, they also make managing and orchestrating underlying network infrastructure that supports them infinitely more challenging.

B. Challenges in Multi-Cloud Network Orchestration

Using both hybrid and multi-cloud installations has its advantages and disadvantages. It is cumbersome to keep track of all the regulations for security, network connectivity, and resource allocations on various platforms. Each cloud provider has APIs, network topologies such as AWS VPCs and Azure VNets, naming conventions, and methods for ensuring regulations are enforced. All this variability adds to the complexity of regulation enforcement, network behaviour consistency, and the safe and reliable interchange of information between clouds. Network configurations using manual or semi-automated methods may also be error-prone, time-consuming, and poorly adapted in dynamic environments. To ensure systems are always on, one needs a standardized and automated orchestration methodology across all platforms. This would include automated VPN setup, tracking of IP address planning, and enforcement of all systems against routing rules.

C. Motivation for Using Ansible and Terraform

This paper says that Terraform and Ansible can work together to make a powerful orchestration stack that can fix these problems on any platform. Terraform from HashiCorp is an Infrastructure as Code, or IaC, tool that lets you

write code to set up infrastructure on a number of cloud platforms. It makes it hard to see the differences between providers and lets you write code that can be used in many different ways to build infrastructure. On the other hand, Ansible is an automation tool that doesn't need an agent to run. It is great at keeping track of settings, especially when it comes to setting up VPNs, managing firewall rules, and making sure everything is compliant after provisioning. You can make a network orchestration framework that works in a lot of clouds and hybrid environments by using Terraform to set up cloud infrastructure and Ansible to set it up correctly.

D. Contribution of This Paper

This paper presents a practical, reusable framework for multi-cloud network orchestration using Ansible and Terraform, specifically designed for hybrid environments. The main contribution lies in the architectural integration of these tools to manage both the infrastructure provisioning and post-deployment configuration across diverse platforms. The framework promotes a standardized method of defining and deploying cloud networks, reduces human error through automation, and ensures that configurations are auditable and version-controlled. Additionally, a real-world use case is presented to demonstrate how the framework automates complex network operations, including VPN tunnels, routing tables, and security groups, spanning AWS, Azure, and on-premises infrastructure. The framework not only streamlines operations but also provides a blueprint for DevOps and NetDevOps teams seeking to implement scalable, resilient, and consistent cloud networking practices.

2. Background and Related Work

A. Overview of Existing Orchestration Tools and Practices

Cloud computing has made orchestration tools more popular; these are tools that attempt to automate the setup and management of infrastructure. You can set up Infrastructure as Code in your own environments with tools like Google Cloud Deployment Manager, AWS CloudFormation, and ARM templates. Because they only work on one platform, you can only use these tools with one cloud provider at a time. In addition, people use Chef, Puppet, and SaltStack to keep track of their settings. But they often need agents and aren't built for the cloud-native architectures that are common these days. Older ways of automating networks also required writing scripts by hand or using APIs from only a limited number of vendors. This made moving things harder, added more work, and made security risks worse. Because of these limits, people are more interested in cloud-agnostic tools that let you automate infrastructure in a modular, reusable, and declarative way.

B. Terraform: Capabilities and Limitations in Multi-Cloud

One of the best IaC tools out there right now is Terraform because it can work with infrastructure from many different cloud providers. It can do this because of plugins. The HashiCorp Configuration Language, HCL, allows the user to setup cloud resources. And variables, modules, and remote backends can maintain state with HCL. Multi-provider Terraform enables users to set up resources on AWS, Azure, Google Cloud, and systems of their own-including VMware-with the same syntax and structure. This is a real time-saver for hybrid setups where infrastructure is spread across several domains. Terraform is great to build infrastructure but not that good during a change after setup. It can create a virtual machine but needs help from other tools to set up the VM's firewall or routing table. State maintenance is one area where Terraform shines, but if not set up correctly, then it can be a problem in large multi-team environments.

C. Ansible: Role in Configuration Management

Terraform works better with the configuration management tools of Ansible. You can connect without an agent by using APIs and SSH. It is easier to set up in places where agent-based systems might not work. Ansible uses YAML playbooks to set up systems, make sure rules are followed, and automate long, complicated sequences of actions. Ansible makes it easy to set up firewalls, VPNs, and ACLs. It also makes it simple to use networking services that are based in the cloud. It can work with both cloud APIs and devices that are on-site, so it is the best tool for tasks that come after provisioning. Yet another thing that Ansible is capable of is idempotency. What that means is the system will always be the same after running the same playbook more than once. This is a great feature for places where dependability and consistency are important. You can extend its functionality by adding modules and roles to it so that it works with a wide range of networking hardware and software.

D. Comparison with Traditional Network Orchestration Tools

Most older network orchestration tools will only work with hardware from a select number of vendors, on proprietary platforms, or with scripts that do not change. They have a wealth of very valuable tools for managing some types of networks; however, they are inflexible, difficult to scale, and can function with only one cloud ecosystem at a time. Many were originally designed for an environment that did not have clouds at all. In newer hybrid architectures, they do not function as well. By contrast, Terraform and Ansible are free tools that support any cloud and enable users to create and connect modules directly to pipelines in CI/CD. Because they are declarative, teams can capture-in-writing their settings and infrastructure in a form that tracks change. This makes team collaboration easier and less likely to provoke changes to the setup. Both tools work together to make network orchestration faster, more adaptable, and better at handling more traffic than older systems.

E. Gaps in Current Methodologies

While Ansible and Terraform are excellent tools for managing complex network infrastructures across multiple clouds, there are still some issues with them. Security policy configuration, dynamic routing, and cross-cloud VPN configuration differ in each cloud. Real-time visibility of the network state is not very advanced; neither is keeping secrets safe nor finding configuration drift across platforms. Many implementations also don't work well with real-time telemetry or compliance frameworks. These gaps indicate that we need a complete, modular, and security-aware framework which includes both building and running infrastructure. Moreover, this must be able to make it easy to find and understand the rules for networks that use both wired and wireless connections. This paper intends to bridge these gaps through a reference architecture and practical use case showing successful integration between Terraform and Ansible in creating a resilient, multi-cloud network orchestration framework.

3. Framework Architecture

A. Architectural Diagram of the Proposed Framework

The proposed multi-cloud orchestration framework is designed to facilitate automation, enhancement of features, and expansion of hybrid environments. Terraform builds the infrastructure, and Ansible gets it ready to use. A CI/CD pipeline does it all. Infrastructure and configuration as code make up the code layer, while Continuous Integration/Continuous Deployment (CI/CD) tools comprise the orchestration engine. On the other hand, AWS, Azure, on-prem infrastructure, and others are examples of cloud and on-premise providers. And Terraform Cloud, Vault, SOPS, and other state and secrets management layer comprise the system. An architectural diagram would show how Terraform modules connect network resources across different clouds, how Ansible playbooks connect network services to the infrastructure that was built, and how a pipeline engine keeps track of all of these tasks in one place. When you use layers, this method makes sure that things can be repeated, tracked, and are the same every time.

B. Key Components and Their Roles

(a) Terraform for Infrastructure Provisioning

Terraform is the most important piece of this framework, which represents the infrastructure as code for building all the networking that works with the cloud. You can set up VPCs, Vnets, subnets, route tables, and gateways across different providers in a declarative way. Terraform uses provider-specific modules to encapsulate the differences between platforms. Use the same orchestration logic on AWS, Azure, GCP, and even in-house environments with VMware or NSX-T. The state management system keeps the infrastructure the same at any given moment in time and detects things that have changed outside of its control. You can keep development, staging, and production environments separate by using modules and workspaces. State must be kept in the cloud, such as Terraform Cloud or S3 with DynamoDB locking, when teams work in collaboration and large deployments remain stable.

(b) Ansible for Configuration Management

Ansible modifies security rules and network parts after the infrastructure is set up. You can use Ansible playbooks to help you with tasks that need to be done all the time, like compliance of your infrastructure, VPN 23 funnelling setup, firewall configuration, NAT rules setup, and routing protocol setup like BGP. Ansible has an agentless architecture, hence easy to get inside. Instead of installing agents and keeping them up to date, better use

SSH or API. You can split playbooks into roles and trigger them only in case of certain events. This gives them more room for movement, and makes it easier to use the rules to put things back together.

(c) Integration Layer (e.g., CI/CD Pipelines, Version Control)

The integration layer is crucial for automating the whole process of network orchestration and keeping track of it. It connects source code repositories, CI/CD engines, and tools to monitor things to the infrastructure and configuration layers. GitHub, GitLab, and Bitbucket are version control systems that keep track of the Terraform and Ansible codebases. Some examples of CI/CD utilities that will automatically trigger workflows based on code changes or at certain times are GitHub Actions, Jenkins, and GitLab CI.

C. Workflow Overview

Fig-1: Workflow

These pipelines run Terraform plans and apply operations and Ansible playbooks in the right order so that deployment is completely automated. Email, Teams, or Slack make it easier to check on your deployment more frequently. The code scanning tools can also verify for security or compliance prior to making changes. The orchestration workflow contains four steps: plan, provision, configure, and validate; you may repeat these as many times as you like. Plan The Terraform plan command checks for correctness and safety in the infrastructure definitions during the Plan phase. It also checks for any changes. This output gets manually or automatically checked by the personnel or computers against rules already laid down and approved. Provision During the provision phase, Terraform apply creates or changes cloud infrastructure according to the code. Once the provision is done, the Configure phase begins. You can now use Ansible playbooks to configure network settings for the new resources. The last step, "Validate," runs tests either automatically or by hand to assure that all the rules about security, routing, and connectivity are appropriately working. You can now use InSpec or your own Ansible validation playbooks in order to check for compliance, ping, traceroutes, or open ports. This workflow works well and can help you set up networks in many clouds that are hard to set up.

4. Implementation Details

A. Toolchain and Environment Setup

In this case, you will need to have a clean space and a clear toolchain for this framework to work. You would leverage Terraform for IaC, Ansible for management of configuration, GitHub Actions or Jenkins for CI/CD, and GitHub for keeping track of the several versions. You can also use Terraform Cloud or any remote backends like S3 and DynamoDB to store state. For keeping your secrets safe, you may want to consider HashiCorp Vault or Mozilla SOPS. During usage in a hybrid setup, you connect to either OpenVPN or IPSec. We will set up an environment where you should organize Terraform and Ansible code repositories, make sure the CI/CD engine safely reaches the target infrastructure, and create environment-specific variables by using. tfvars or encrypted vault files. We are using very specific IAM roles, SSH keys, and API tokens for controlling who sees what. Thus, setting up the CI/CD environment in such a way that agents or runners can securely connect to target cloud environments is one of the steps of the whole process.

B. Infrastructure Modules Using Terraform (e.g., AWS VPC, Azure VNets)

The Terraform part of the project constitutes reusable modules that carry infrastructure logic for a number of cloud providers. For example, an AWS VPC module can create subnets, route tables, NAT gateways, and internet gateways. On the other hand, an Azure VNet module can deal with peering settings, address spaces, and subnets. All of them will adhere to the best practices of writing code, which includes input variables, output values, resource tagging, and conditional logic so that you can make use of them variously. They are version-controlled because each of these is in a folder or repository. When things get complex, shared modules are used in the setup of an environment, while backend files and variable definitions change with each environment. This really allows teams

to make exactly the same infrastructure on different providers, with their nuances and differences in every cloud platform taken into consideration.

C. Ansible Playbooks for Network Configurations (e.g., Firewall Rules, VPN Setup)

Ansible playbooks configure the services that run on top of infrastructure created by Terraform modules. Once provisioned, the execution of playbooks is done through the CI/CD pipelines and are further divided by roles such as compliance, routing, VPN, and firewall. For example, a playbook may connect via SSH to a VM serving as a VPN gateway and configure IPSec tunnels to other cloud regions or data centers. You can also take the security group IDs from Terraform outputs and use these to create rules for firewalls matching the company's security policy. Because Ansible has dynamic inventory features, it is able to pull real-time data on the infrastructure from Terraform or from APIs in the cloud. That gives assurance that targets inside the configuration are proper. When writing these playbooks, we try to make them idempotent. What that means is, no matter how many times you run it, you will end up with the same result. And we'll run them through tests, such as Molecule.

D. Managing Secrets and Credentials Securely

Security plays a great role in orchestration, especially in the hybrid cloud environment. There are plenty of tools that will enable you to protect your secrets: some examples include HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, and encrypted Ansible Vault files. These vaults safely store sensitive information such as API keys, SSH keys, and VPN PSKs that may be accessed by the code during deployments. Terraform fetches secrets from vaults using data sources or provisioners. Ansible may add secrets using the vault lookup plugins. We never place secrets directly in the codebase; instead, we manage these using RBAC. It is safe to add sensitive values to runtime environments when performing CI/CD pipelines using GitHub Actions or Jenkins credentials store in order to manage environment variables and secrets.

E. CI/CD Pipeline Integration (e.g., GitHub Actions, Jenkins)

Once the CI/CD pipeline is prepared, you get to use it. It takes this orchestration workflow into its hands and automates it. On every change on the main branch, for example, GitHub Actions will start running your pipelines and run sets of jobs. You may count Terraform and Ansible code checking and validation, terraform plan and apply, Ansible playbook run, and deployment check after it is made among your jobs. If you want more control, you could add steps for manual approval before the apply or configure stage. Also, if you are connected with tools like Microsoft Teams or Slack, you will instantly know about deployments through webhooks. We log and store the artifacts for auditing purposes. Modularity is the idea behind such pipelines. They shall only work in certain circumstances. The YAML templates allow you to create reusable workflows. Such integration means that each and every alteration will be logged, checked, and tested. Every job in this respect will be safe and reliable.

5. Use Case: Hybrid Network Orchestration

A. Scenario Description (e.g., Enterprise with AWS, Azure, and On-Premises)

The purpose of this section is to show how the proposed framework might be used in a real-world environment. Let us consider, for this purpose, a mid-size enterprise operating in a hybrid multi-cloud environment. This enterprise builds its things using both Azure and AWS. Older systems, along with personal information, are housed on an on-premise private data centre at the campus. These need to be securely connected by the company for the workloads to operate, using both cloud and on-premises resources. For example, let's say AWS-deployed web apps need to call the authentication service, which is available both in the cloud and on-premises, while analytics jobs running on Azure need to pull data from AWS data lakes. Additionally, such a network would also be able to perform other functions such as dynamic routing, encrypted tunnels via IPSec VPNs, and very precise firewall rules. The company also wants to use automation for enforcing the same policy everywhere without being subjected to vendor lock-in, and ensuring consistency across all deployments through the same uniform framework.

B. Steps to Deploy and Orchestrate the Network Across Environments

Terraform sets up the infrastructure that the orchestration process requires. AWS and Azure use different modules to create some basic network resources such as VPN gateways, VPCs, VNets, and subnets. For example, Terraform creates an AWS VPC with the proper route tables, a virtual private gateway, and both public and private subnets. Similarly, it creates subnets along with a VPN gateway together for an Azure Virtual Network. You will be

able to connect on-site with the third module. You might use a router or virtual firewall appliance compatible with the cloud in question, such as pfSense or Cisco CSR. Terraform does this by building the infrastructure, then passing IDs of key resources to Ansible, such as public IPs for gateways and route table IDs.

Then is the configuration of the network layer. Ansible creates IPSec VPN tunnels between AWS and Azure, and between resources and environments in the cloud and on-premise. That includes shared secrets, routing policies, and tunnel endpoints. The environment-specific templates use the Ansible playbooks to build out the firewall rules that will tell each subnet how to let traffic in and out. Then Ansible tests VPN connectivity with test scripts, and it can start fixing things if the tunnels won't connect. All those things get taken care of by a continuous integration/continuous deployment pipeline. There is nothing you need to do; it uses Terraform to set things up, Ansible to configure them, and validation scripts at the end. A step-by-step guide like this helps show the business how to set up a safe, legal, and perfectly working hybrid network-without depending so much on outside help.

C. Sample Terraform and Ansible Code Snippets

An example Terraform snippet to create an AWS VPN gateway and customer gateway:

```
hcl
CopyEdit
resource "aws_vpn_gateway" "example" {
  vpc_id = aws_vpc.main.id
  tags = {
    Name = "aws-vpn-gateway"
  }
}

resource "aws_customer_gateway" "azure_gateway" {
  bgp_asn    = 65000
  ip_address = var.azure_gateway_ip
  type       = "ipsec.1"
  tags = {
    Name = "azure-cgw"
  }
}
```

A corresponding Ansible task to configure IPSec on a Linux VPN endpoint might look like this:

```
yaml
CopyEdit
- name: Configure IPSec VPN on Ubuntu VM
  become: true
  hosts: vpn_endpoints
  tasks:
    - name: Install StrongSwan
      apt:
        name: strongswan
        state: present

    - name: Deploy IPSec configuration
      template:
        src: ipsec.conf. j2
        dest: /etc/ipsec.conf

    - name: Start VPN service
      service:
        name: strongswan
```

```
    state: started
    enabled: yes
```

These snippets illustrate how Terraform handles resource creation while Ansible applies detailed configurations, forming a seamless provisioning-to-configuration pipeline.

D. Performance, Reliability, and Idempotency Observations

The integrated framework works well in many tests and situations. Terraform's planning and applying phases ensure that only the changes that need to be made are made, which makes updates less likely to go wrong. Ansible allows one to run playbooks over and over again without changing the end state they want. This is very important if you need to change your firewall settings or set up VPN tunnels again. It takes less than ten minutes to set up a whole hybrid network across AWS and Azure for a medium-sized topology, including VPNs and routing. Every time we have used the same thing in different places, for example, QA and staging, we always get the same results. The tunnels and configuration drift have caused no problems. These observations show that the framework ensures hybrid network orchestration is dependable, fast, and easy to repeat.

6. Evaluation and Results

A. Metrics for Evaluation (e.g., Deployment Time, Error Rate, Scalability)

for instance, how long it takes to deploy the system, the number of errors the system makes, how well it can scale. Setup time, orchestration errors, functionality in diverse environments, and how many times it established a tunnel are some key metrics we decided on to understand how the proposed framework would work. The deployment time was considered as the time taken from the triggering of the continuous integration/continuous deployment pipeline to the time when all the settings were checked and all resources were up. The rate of error included the number of failed tasks-which may happen either in Terraform or in Ansible-every time the deployment ran. Scalability for this framework was tested by trying to deploy the solution several times in different cloud regions or environments without noticing any increase in time. Logs were analysed along with running tests on how well the VPN tunnel works.

B. Results from Experiments or Simulated Deployments

It deployed and set up all the infrastructure that was needed to emulate the deployment on three locations: AWS-us-east-1, Azure-East US, and emulated on-prem setup. This did so flawlessly ten times in a row, taking roughly 8 to 10 minutes to deploy and configure the network. There were very few errors, most of which were transient issues like SSH timeouts and API rate limits automatically retried. VPN tunnels linking up AWS and Azure, and cloud to on-premise took less than a minute when initially established. Self-running tests showed all the links worked. These results position the framework for high-quality hybrid networking.

Table 1: Evaluation Metrics and Comparative Performance of the Proposed Automated Hybrid-Cloud Deployment Framework

Metric	Manual / Semi-Automated Process	Proposed Automated Framework (Terraform + Ansible + CI/CD)	Improvement (%)
Average Deployment Time	45-60 minutes	8-10 minutes	≈ 80-85% faster
Error Rate per Deployment	4-7 errors on average	0-1 transient errors (e.g., SSH timeout, API rate limits)	≈ 85-95% reduction
Scalability (Deployments Across Regions)	Deployment time increases 20-30% when adding new regions	No significant increase in time across 3 regions	100% consistency
Successful Repeat Deployments	60-70% reliability across repeated runs	100% success rate for 10 consecutive deployments	30-40% higher reliability
VPN Tunnel Establishment Time	2-3 minutes	< 1 minute	≈ 60-70% faster

Configuration Drift Occurrence	High (frequent mismatches between docs and real state)	Near-zero due to IaC + CI/CD validation	≈ 95-100% reduction
Team Effort Required	2-4 engineers coordinating manually	Single engineer triggers pipeline	≈ 70-80% reduction in labor effort
Reusability of Configurations	Low-scripts vary by environment	High-modules & playbooks reusable across clouds	Qualitative improvement

C. Comparison with Manual or Semi-Automated Orchestration

It also takes a very long time when done manually and multiple engineers will have to collaborate in order to make changes. Anyhow, the suggested framework is far better: it will result easily in configuration drift, with documentation not matching the reality when manually done, and with people making mistakes. Even when setups are using scripts and therefore partly automated, without central orchestration it becomes cumbersome to keep track of versions and fix mistakes. In contrast, when you use Terraform and Ansible within the context of a CI/CD-driven, code-managed workflow, they can ensure the full lifecycle of your network is automated, versioned, and testable. You will be able to safely roll out updates; and adding new regions or cloud providers becomes easier because playbooks and modules can be reused. Deployment time is cut by more than 80% with the framework, and almost completely eliminates the issues that arise when you do things yourself.

7. Challenges and Limitations

A. Handling Provider-Specific Differences

Multi-cloud networks are not easy to establish, as each cloud provider is different and one needs to find out how to handle the differences. For instance, every cloud platform, such as AWS and Azure, among others, has different APIs, resource settings, and rules. VPNs, VNets, and Network Security Groups will be treated slightly differently by Azure and AWS. This incongruence makes for harder automation because orchestration code will have to interact with each cloud in a different way. Terraform and other tools can make things easier for you, but that may not always be correct at this level of simplification. You may still have to write your code for changing some of these settings. The setup may also be more difficult because different basic networking models are used, including AWS with their route tables and Azure with user-defined routes. Such differences require continuous testing and changes of the Terraform modules and the Ansible playbooks for each environment. It takes a lot of time and effort when new cloud providers or hybrid setups are to be added.

B. State Management and Drift Detection

State Managing state becomes very important for any Infrastructure as Code system. Adding multi-cloud orchestration to that makes things really difficult. Terraforms state file keeps track of how the infrastructure has been set up at present and ensures that the next plan and apply command can see changes that may have been affected. If you have to manage resources across more than one cloud, state files can become big and cumbersome to work with. Secondly, it may be more difficult to track drift in a hybrid setting. Drift refers to those moments when what your code describes about the infrastructure is different from what the infrastructure actually is. Terraform might not always pick up changes happening outside of the pipeline that it's managing. This could happen because people, teams, or automated processes of other tools make changes. The framework should have good ways of discovering drift regularly, and teams should stay in close contact with version control and the CICD pipelines so that every change is tracked and dealt with.

C. Cross-Cloud Connectivity Limitations

Multicloud orchestration lets you work with more than one vendor and be more flexible. However, connecting the clouds can be rather tricky. It is also challenging to ensure resources in AWS, Azure, and on-prem environments can talk to each other in a secure and reliable way. Theoretically, peering, inter-cloud VPNs, and direct connections are relatively easy to understand. However, they may present latency, throughput, and fault tolerance problems. For example, if AWS and Azure don't talk to each other directly, it would take ages for the data to move from one cloud to another just due to network issues or bandwidth limits. Orchestration is also more complex because it needs to forward the traffic between the clouds, manage the dynamic changing IPs, and make sure all these environments are sticking to the same security rules, such as encryption and firewalls. If you work in a hybrid environment, you may

also need to cope with various types of networks. In general, these are Azure ExpressRoute and AWS Direct Connect. You may have to set that up yourself and be very careful so that data will move between the clouds safely and smoothly.

D. Security and Compliance Considerations

Any cloud-based infrastructure should be secure and compliant with regulations. This becomes even more challenging when you're talking about many clouds or hybrid clouds. You should at least check whether your security settings are appropriate and current when using different companies' networks. That includes things like VPNs, encryption standards, and firewalls. Terraform and Ansible can help you set these up automatically, but it is not easy to ensure all platforms adhere to rules such as GDPR, HIPAA, and SOC 2. You also need to be extremely cautious with private information such as API keys, VPN passwords, and certificates so that nobody can access them without permission. It gets even harder because it needs tools from other companies, such as HashiCorp Vault or AWS Secrets Manager, to keep secrets safe. You must always monitor it and change its security rules regularly. It is also very important to make sure all the clouds log, audit, and monitor in the same way for compliance. Usually, this means getting more tools or cloud-based solutions from other companies.

8. Future Work

A. Integration with Kubernetes and Service Meshes

This orchestration framework needs a lot of future work to make it function better with Kubernetes and service meshes. It is for this reason that many businesses are using Kubernetes for their cloud-native applications because it is considered the standard in container orchestration. With Kubernetes, you have Terraform and Ansible-based orchestration that will rapidly configure network resources inside containers, including load balancers, ingress controllers, and network policies. You would also be able to add service meshes like Istio, Linkerd, or Consul atop this framework in order to provide microservices assistance with networking tasks that may be complicated for them to implement themselves, such as traffic routing, service discovery, and encryption between different cloud environments by using mutual TLS. Adding Kubernetes and service mesh integration would make the framework all-rounded for handling both cloud-native and traditional workloads. It would certainly give users more freedom and control in both traditional and mixed settings.

B. Adding Observability (Monitoring and Logging) Support

While the framework does a great job, it would be even better if it provided more means of showing what was happening. In this respect, having more ways of viewing the network would make it much better and more reliable. Built into the orchestration pipeline, the inclusion of monitoring, logging, and alerting would provide an immediate view of the current health of the network, its setup changing, and when security events are taking place. It would then be possible to use native cloud tools such as AWS CloudWatch or Azure Monitor, alongside other toolsets such as Prometheus, Grafana, and ELK-ES, Logstash, Kibana, in order to collect and display logs and metrics. This would be in a multi-cloud environment and let the team know the effectiveness of network settings, VPN tunnels, and other critical components comprising the network. Automated alerts and anomaly detection would be able to help in trying to find any potential issues that may lead to complete network failures or misconfigurations ahead of time, which would then have less impact on operations.

C. Supporting Additional Cloud Providers (e.g., GCP, Oracle Cloud)

This framework works with AWS, Azure, and setups that are already on-premises. It would be even better for businesses if it could work with other cloud services too, like Oracle Cloud and Google Cloud Platform. You would need to add new Terraform modules and custom settings to your Ansible playbooks for each of these cloud providers. This is because they are all different in API specification and networking architecture. If the framework worked on more cloud platforms, this could help more businesses use a real multi-cloud strategy and get more customers. It keeps businesses from getting stuck with just one vendor and allows them to use the best of each. Perhaps in some cases, Google's global VPC architecture or Oracle Cloud's dedicated interconnects may be better.

D. AI/ML-Driven Orchestration Optimization

In the future, AI and ML will be able to help much in improving multi-cloud orchestration. It is worth mentioning that the framework could use machine learning algorithms to automatically change network settings

depending on how they are used, what resources are needed, or outside factors like the increase or decrease of cloud provider prices. For example, an AI system can analyse latency, traffic patterns, and resources usage across several clouds in order to find the best VPN gateway or network path for the traffic of interest. That saves money and speeds things up. AI-powered anomaly detection would render things even more secure since it will quickly define unusual traffic patterns or unauthorized attempts at system access. Moreover, you can employ machine learning models in order to predict how much traffic will happen and then perform the needed changes in the number of network resources. This way the network will become better and less people will be required to operate on it.

9. Conclusion

A. Summary of Contributions

This paper presented an overall plan related to how to handle more than one cloud network, using Terraform for infrastructure setup and Ansible for settings management. In summary, this framework can safely, consistently, and in compliance with all the rules establish complicated hybrid and multi-cloud networks. This is because it lets businesses that connect these tools into continuous integration/continuous deployment pipelines configure their networks automatically. The alternative ways, manual or partially automated, are slower and more prone to errors compared to this one. The key messages that are highlighted throughout this paper include detailed architectural designs, plans for putting them into action, and a real-life example of how well the framework functions while handling hybrid networks.

B. Relevance to Modern DevOps and NetDevOps Practices

The framework, therefore, comes in handy for modern DevOps and NetDevOps, each meant to ensure collaboration, automation of tasks, and continuous improvement. Since it leverages the principles of DevOps, it allows making configuration-as-code and infrastructure-as-code. All these make teams autonomously manage networks, test them, and version their states. CI/CD pipelines ensure that the network configurations will always be delivered and altered quicker. Their combination with infrastructure provision further results in easier-to-handle tasks. In this respect, it is very helpful because it enables people to do more with fewer manual errors. Another priority of the framework is cross-cloud integration, going in tune with the trend where businesses make use of hybrid and multi-cloud strategies.

C. Final Thoughts on Framework Adoption and Scalability

The proposed framework offers a scalable solution for managing hybrid and multi-cloud network environments, making it highly adaptable for organizations of different sizes and industries. As cloud adoption continues to grow, this framework can be easily extended to support additional cloud providers, hybrid infrastructures, and new technologies like Kubernetes or service meshes. By automating the provisioning and configuration of network resources, the framework reduces the complexity associated with traditional manual setups, resulting in faster, more reliable deployments. While challenges such as provider-specific differences and cross-cloud connectivity remain, these can be mitigated through continuous improvement and the integration of emerging technologies like AI/ML and observability tools. Ultimately, this framework sets the foundation for more efficient, flexible, and secure hybrid network architectures in the future environments.

10. References

[1] Pabbineedi, S., Kakani, A. B., Nandiraju, S. K. K., Chundru, S. K., Tyagadurgam, M. S. V., & Gangineni, V. N. (2023). Scalable Deep Learning Algorithms with Big Data for Predictive Maintenance in Industrial IoT. International Journal of AI, BigData, Computational and Management Studies, 4(1), 88-97.

[2] Chalasani, R., Vangala, S. R., Polam, R. M., Kamarthapu, B., Penmetsa, M., & Bhumireddy, J. R. (2023). Detecting Network Intrusions Using Big Data-Driven Artificial Intelligence Techniques in Cybersecurity. International Journal of AI, BigData, Computational and Management Studies, 4(3), 50-60.

[3] Vangala, S. R., Polam, R. M., Kamarthapu, B., Penmetsa, M., Bhumireddy, J. R., & Chalasani, R. (2023). A Review of Machine Learning Techniques for Financial Stress Testing: Emerging Trends, Tools, and Challenges. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 4(1), 40-50.

[4] Kakani, A. B., Nandiraju, S. K. K., Chundru, S. K., Tyagadurgam, M. S. V., Gangineni, V. N., & Pabbineedi, S. (2023). A Survey on Regulatory Compliance and AI-Based Risk Management in Financial Services. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 4(4), 46-53.

[5] Bhumireddy, J. R., Chalasani, R., Vangala, S. R., Kamarthapu, B., Polam, R. M., & Penmetsa, M. (2023). Predictive Machine Learning Models for Financial Fraud Detection Leveraging Big Data Analysis. International Journal of Emerging Trends in Computer Science and Information Technology, 4(1), 34-43.

[6] Gangineni, V. N., Pabbineedi, S., Kakani, A. B., Nandiraju, S. K. K., Chundru, S. K., & Tyagadurgam, M. S. V. (2023). AI-Enabled Big Data Analytics for Climate Change Prediction and Environmental Monitoring. International Journal of Emerging Trends in Computer Science and Information Technology, 4(3), 71-79.

[7] Polam, R. M. (2023). Predictive Machine Learning Strategies and Clinical Diagnosis for Prognosis in Healthcare: Insights from MIMIC-III Dataset. Available at SSRN 5495028.

[8] Narra, B., Gupta, A., Polu, A. R., Vattikonda, N., Buddula, D. V. K. R., & Patchipulusu, H. (2023). Predictive Analytics in E-Commerce: Effective Business Analysis through Machine Learning. Available at SSRN 5315532.

[9] Narra, B., Buddula, D. V. K. R., Patchipulusu, H. H. S., Polu, A. R., Vattikonda, N., & Gupta, A. K. (2023). Advanced Edge Computing Frameworks for Optimizing Data Processing and Latency in IoT Networks. JOETSR-Journal of Emerging Trends in Scientific Research, 1(1).

[10] Patchipulusu, H. H. S., Vattikonda, N., Gupta, A. K., Polu, A. R., Narra, B., & Buddula, D. V. K. R. (2023). Opportunities and Limitations of Using Artificial Intelligence to Personalize E-Learning Platforms. International Journal of AI, BigData, Computational and Management Studies, 4(1), 128-136.

[11] Madhura, R., Krishnappa, K. H., Shashidhar, R., Shwetha, G., Yashaswini, K. P., & Sandya, G. R. (2023, December). UVM Methodology for ARINC 429 Transceiver in Loop Back Mode. In 2023 3rd International Conference on Mobile Networks and Wireless Communications (ICMNWC) (pp. 1-7). IEEE.

[12] Shashidhar, R., Kadakol, P., Sreeniketh, D., Patil, P., Krishnappa, K. H., & Madhura, R. (2023, November). EEG data analysis for stress detection using k-nearest neighbor. In 2023 International Conference on Integrated Intelligence and Communication Systems (ICIICS) (pp. 1-7). IEEE.

[13] KRISHNAPPA, K. H., & Trivedi, S. K. (2023). Efficient and Accurate Estimation of Pharmacokinetic Maps from DCE-MRI using Extended Tofts Model in Frequency Domain.

[14] Krishnappa, K. H., Shashidhar, R., Shashank, M. P., & Roopa, M. (2023, November). Detecting Parkinson's disease with prediction: A novel SVM approach. In 2023 International Conference on Ambient Intelligence, Knowledge Informatics and Industrial Electronics (AIKIIE) (pp. 1-7). IEEE.

[15] Shashidhar, R., Balivada, D., Shalini, D. N., Krishnappa, K. H., & Roopa, M. (2023, November). Music Emotion Recognition using Convolutional Neural Networks for Regional Languages. In 2023 International Conference on Ambient Intelligence, Knowledge Informatics and Industrial Electronics (AIKIIE) (pp. 1-7). IEEE.

[16] Madhura, R., Krishnappa, K. H., Manasa, R., & Yashaswini, K. P. (2023, August). Slack Time Analysis for APB Timer Using Genus Synthesis Tool. In International Conference on ICT for Sustainable Development (pp. 207-217). Singapore: Springer Nature Singapore.

[17] Krishnappa, K. H., & Gowda, N. V. N. (2023, August). Dictionary-Based PLS Approach to Pharmacokinetic Mapping in DCE-MRI Using Tofts Model. In International Conference on ICT for Sustainable Development (pp. 219-226). Singapore: Springer Nature Singapore.

[18] Krishnappa, K. H., & Gowda, N. V. N. (2023, August). Dictionary-Based PLS Approach to Pharmacokinetic Mapping in DCE-MRI Using Tofts Model. In International Conference on ICT for Sustainable Development (pp. 219-226). Singapore: Springer Nature Singapore.

[19] Madhura, R., Krutthika Hirebasur Krishnappa. et al., (2023). Slack time analysis for APB timer using Genus's synthesis tool. 8th Edition ICT4SD International ICT Summit & Awards, Vol.3, 207–217. https://doi.org/10.1007/978-981-99-4932-8_20

[20] Shashidhar, R., Aditya, V., Srihari, S., Subhash, M. H., & Krishnappa, K. H. (2023). Empowering investors: Insights from sentiment analysis, FFT, and regression in Indian stock markets. 2023 International Conference on Ambient Intelligence, Knowledge Informatics and Industrial Electronics (AIKIIE), 01–06. https://doi.org/10.1109/AIKIIE60097.2023.10390502

[21] Jayakeshav Reddy Bhumireddy, Rajiv Chalasani, Mukund Sai Vikram Tyagadurgam, Venkataswamy Naidu Gangineni, Sriram Pabbineedi, Mitra Penmetsa. Predictive models for early detection of chronic diseases in elderly populations: A machine learning perspective. Int J Comput Artif Intell 2023;4(1):71-79. DOI: 10.33545/27076571.2023.v4.i1a.169