Technology
Arts Sciences
TH Köln

cimt

Terraform and CloudFormation: A Comparative Study of Infrastructure as Code on AWS

RESEARCH PROJECT

by

Matthew Esosa Igbinehi

submitted as a requirement for

MASTER OF SCIENCE (M.SC.)

at

TH KÖLN UNIVERSITY OF APPLIED SCIENCES
INSTITUTE OF COMPUTER AND COMMUNICATION TECHNOLOGY

Course of Studies

COMMUNICATION SYSTEMS AND NETWORKS

First supervisor:	Prof. Andreas Grebe TH Köln University of Applied Sciences
Second supervisor:	Hrvoje Husic Cimt ag Services

Cologne, November 2023

Contact details:

Matthew Esosa IGBINEHI
Grüngürtelstr. 122
50996 Köln
matthew_esosa.igbinehi@smail.th-koeln.de

Prof. Dr. Andreas GREBE
Cologne University of Applied Sciences
Institute of Computer and Communication Technology
Betzdorfer Straße 2
50679 Köln
andreas.grebe@th-koeln.de

Hrvoje HUSIC
Cimt ag Services
Oskar-Jäger-Straße 170
50825 Köln
Hrvoje.Husic@cimt.de

Abstract

The emergence of cloud computing with leading cloud providers, AWS, Microsoft, and Google, offering unmatched scalability and flexibility, has revolutionised how companies provision and configure their IT infrastructures. However, manually provisioning and managing these cloud resources remains challenging, especially for complex infrastructure. Infrastructure as Code (IaC) has emerged as a pivotal solution to address this issue. This research aims to comprehensively examine and compare the effectiveness of two popular IaC tools, Terraform and AWS CloudFormation, in automating cloud infrastructure provisioning within the AWS environment. The study investigates each tool's key features and functionality, focusing on the AWS environment.

The research methodology encompassed a review of existing literature, practical implementation of a case study using both tools and a subsequent comparison. The primary basis of comparison was performance, developer experience, key features and maintainability.

The findings indicate that both tools are proficient in provisioning and configuring cloud infrastructure and delivering a commendable developer experience. Terraform delivered a better performance, characterised by quicker infrastructure provisioning and destruction operations. On the other hand, AWS CloudFormation provided more detailed documentation and a faster learning experience, especially for developers already familiar with the AWS cloud environment. Furthermore, it recommends opting for Terraform for large and complex infrastructures where modularity using Terraform modules would be a great advantage in infrastructure management and for scenarios involving multi-cloud environments or where using more mature tools is deemed essential.

Acknowledgements

I want to express my sincere gratitude to the individuals and organisations that have contributed to the successful completion of this research.

First and foremost, I extend my most profound appreciation to Hrvoje Husic, my research supervisor at Cimt Ag Services for giving me the opportunity to carry out this research. His guidance and support throughout the entire research process have been invaluable. The opportunity to collaborate with Cimt Ag Services provided a practical and enriching experience that significantly enhanced the quality of this research.

I am also grateful to Prof. Dr. Andreas Grebe from the Institute of Computer and Communication Technology at Cologne University of Applied Sciences. His insightful feedback and academic guidance played a crucial role in shaping the direction of this research.

Special thanks go to Cimt Ag Services, for providing access to their AWS platform. This collaboration facilitated the execution of experiments and the collection of data, enriching the depth and applicability of the research.

Lastly, I want to thank my family and friends for their unwavering encouragement and understanding during this demanding academic journey.

This research would not have been possible without the collective support of these individuals and organisations. Thank you for being a part of this academic endeavour.

Contents

List of Figures

List of Tables

List of Abbreviations

VM	**V**irtual **M**achine
CMS	**C**ontent **M**anagement **S**ystem
IAM	**I**dentity and **A**ccess **M**anagement
AWS	**A**mazon **W**eb **S**ervices
NIST	**N**ational **I**nstitute of **S**tandard and **T**echnology
API	**A**pplication **P**rogramming **I**nterface
SDK	**S**oftware **D**evelopment **K**it
CLI	**C**ommand-**L**ine **I**nterface
NAT	**N**etwork **A**ddress **T**ranslation
PaaS	**P**latform as a **S**ervice
SaaS	**S**oftware as a **S**ervice
IaaS	**I**nfrastructure as a **S**ervice
IaC	**I**nfrastructure as **C**ode
HCL	**H**ashicorp **C**onfiguration **L**anguage
RDS	**R**elational **D**atabase **S**ervice

Chapter 1

Introduction

The emergence of cloud computing has significantly shifted how companies manage and provision their IT infrastructures. AWS, Microsoft, and Google led the 2022 edition of Gartner's Cloud Magic Quadrant for Cloud Infrastructure and Platform Services (CISP), with AWS holding the top spot for the 12th year running [1][2]. Even though these cloud providers provide unmatched scalability and flexibility for implementing various infrastructure solutions, managing these resources is still a crucial and challenging undertaking. Infrastructure as Code (IaC), which automates cloud infrastructure provisioning and configuration, has emerged as a key method to address this issue.

Terraform and AWS CloudFormation, two well-known infrastructure automation tools, have demonstrated a high level of proficiency in enabling infrastructure as code configuration, provisioning and management. In the "Flexera State of the Cloud Report 2022," it is said that 50% of enterprises have already included AWS CloudFormation into their infrastructure operations, and 20% are actively intending to do so. Similarly, 30% of firms have adopted Terraform, and 20% more strongly consider doing the same [3].

The primary objective of this research is to comprehensively examine, evaluate, and draw a comparative analysis of the effectiveness of these two IaC tools, Terraform and AWS CloudFormation, in automating the provisioning of cloud infrastructure within the AWS environment. Our inquiry aims to address fundamental questions revolving around the effectiveness of these IaC tools in automating and managing cloud infrastructure.

The research addresses the following pivotal questions:

- How does AWS CloudFormation compare to Terraform regarding performance, developer experience and maintainability?
- What are the key features and functionalities of Terraform and AWS CloudFormation as Infrastructure as Code tools, and how do these features contribute to the ease of provisioning and managing resources on AWS Cloud?

This research endeavours to address these inquiries using a comparative examination employing the case study approach. Comparative investigations involve evaluating and comparing a phenomenon across domains, subjects, or objects, using quantitative and qualitative methodologies to identify commonalities and variances [4].

The primary aim of this research is to furnish valuable insights intended for cloud professionals and organisations keen on enhancing the management of their cloud infrastructures. The findings are anticipated to be pivotal in facilitating well-informed decisions for selecting the most suitable Infrastructure as Code (IaC) tool tailored to specific use cases and requirements. Consequently, this study will improve our understanding of Terraform and AWS CloudFormation.

Chapter 2

Cloud Computing

There are several definitions of Cloud Computing, but the National Institute of Standards and Technology (NIST) defined it as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [5].

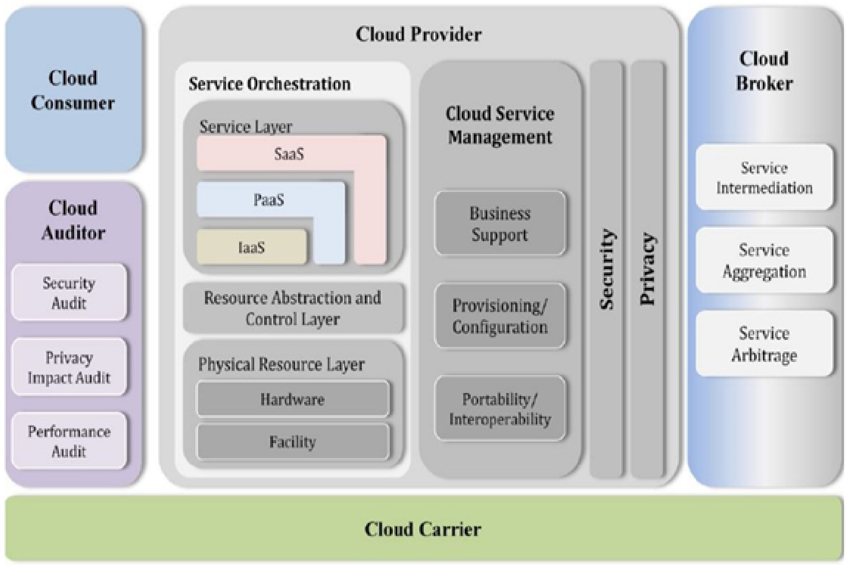


FIGURE 2.1: Cloud Computing Reference Architecture – NIST [6]

2.1 Historical background of cloud computing

In the days before cloud services, hosting a website on the Internet required the installation of physical hardware and servers, even for small businesses and temporary infrastructure. This labour-intensive and time-consuming procedure often required a significant initial investment and a solid understanding of technical requirements. As the number of online businesses increased, so did the need for websites, creating a new market for enterprises to provide hosting and upkeep services. This signalled the start of the revolution in cloud computing.

The history of cloud computing is not complete without a reflection on the vision of computer scientist John McCarthy. He predicted when computing resources would be as easily accessible as utilities. Large mainframe computers were costly in this

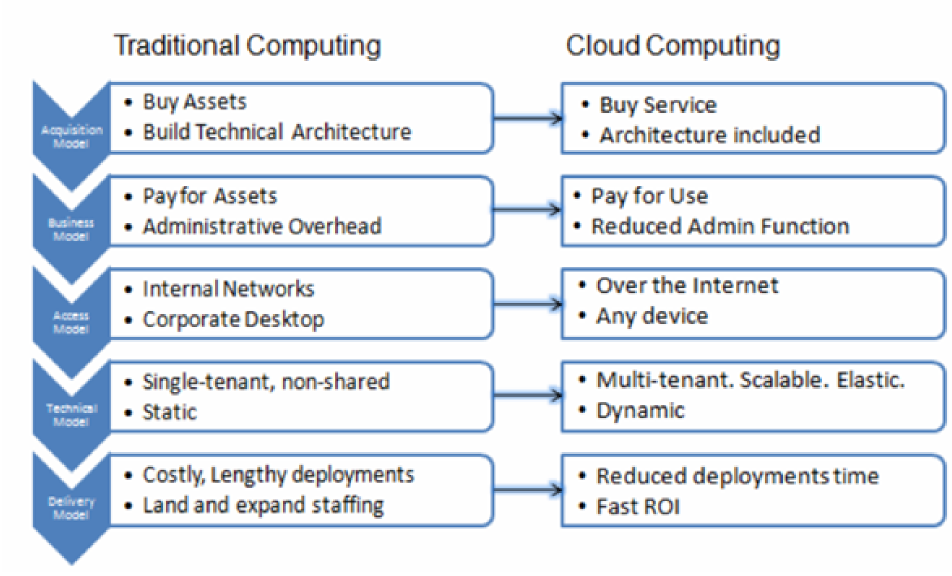


FIGURE 2.2: On-premise computing and cloud computing [7]

era and were utilised mainly by government and research organisations. The utility model in use today for cloud computing grew out of these early concepts. From a technical perspective, cloud computing began to take shape in the 1970s with the advent of virtualisation technology. Virtualization made better use of computing resources by enabling the operation of several operating systems on a single physical machine. Today, there has been a great advancement in this area.

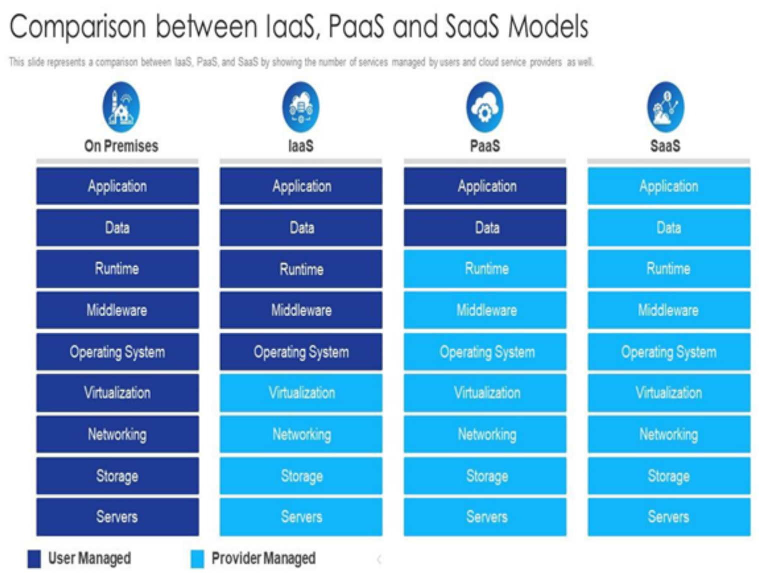
There was a sharp increase in internet-based companies and e-commerce endeavours during the dotcom bubble of the late 1990s. Businesses could obtain processing power and storage through the Internet thanks to the services provided by hosting providers like Salesforce and Amazon Web Services (AWS). This signified the start of cloud services' commercialisation [8]. And on March 14, 2006, Elastic Compute Cloud (EC2), a service that allowed users to rent virtual servers, was introduced by Amazon Web Services (AWS). This was revolutionary because it opened access to easy scalability and cost efficiency for computing resources. Other hyperscalers came on the scene with Google launching its Google App Engine in 2008. In 2010, Microsoft introduced Azure Cloud, its cloud computing platform, expanding the market and escalating competition among cloud service providers.

The decade of the 2010s saw a broad industry acceptance of cloud computing. Businesses adopted the Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS) models. Cloud technology has become a vital part of corporate operations; today, cloud computing is the foundation for digital transformation, facilitating the Internet of Things, Machine Learning, Data Storage, and more. Leading cloud providers are always coming up with new ideas and ways to improve their offerings. Moreover, the rise of edge computing and serverless computing increases the necessity of cloud computing.

The history of cloud computing bears witness to its constant development. Cloud computing is predicted to become increasingly important in determining how businesses, innovation, and other aspects of society develop in the future.

2.2 Models of cloud service

In the context of cloud service models, "service model" refers to the different kinds of cloud services offered to users. These service models specify how much responsibility and control users have over the available resources and underlying infrastructure. The three recognised cloud service models are software as a Service (SaaS, Platform as a Service (PaaS) and Infrastructure as a Service (IaaS)



Comparison between IaaS, PaaS and SaaS Models

This slide represents a comparison between IaaS, PaaS, and SaaS by showing the number of services managed by users and cloud service providers as well.

On Premises	IaaS	PaaS	SaaS
■ Application	■ Application	■ Application	□ Application
■ Data	■ Data	■ Data	□ Data
■ Runtime	■ Runtime	□ Runtime	□ Runtime
■ Middleware	■ Middleware	□ Middleware	□ Middleware
■ Operating System	■ Operating System	□ Operating System	□ Operating System
■ Virtualization	□ Virtualization	□ Virtualization	□ Virtualization
■ Networking	□ Networking	□ Networking	□ Networking
■ Storage	□ Storage	□ Storage	□ Storage
■ Servers	□ Servers	□ Servers	□ Servers

■ User Managed □ Provider Managed

FIGURE 2.3: Cloud service models - comparison between SaaS, PaaS and IaaS models [9]

2.2.1 Software as a Service (SaaS)

A complete application provided as a service to the service user is known as SaaS. All that is required of the service user is to manage users and configure a few application-specific parameters. The provider manages all infrastructure, application logic, deployments, and product or service delivery tasks. Customer relationship management (CRM), enterprise resource planning (ERP), payroll, accounting, and other standard business software are a few of the frequently used SaaS applications. Businesses use SaaS solutions for non-core tasks to avoid hiring personnel to maintain and support the application infrastructure. Instead, they merely use the service as an online resource after paying a subscription fee. NIST defines Software as a Service (SaaS) as the customer being given The ability to use the provider's applications on a cloud infrastructure. The applications can be accessed via a program interface or a client interface like a web browser (for example, web-based email). With the possible exception of restricted user-specific application configuration, the customer has no control over the underlying cloud infrastructure, including the network, servers, operating systems, storage, or even the capabilities of individual applications [10].

2.2.2 Platform as a Service (PaaS)

The goal of PaaS is for developers to focus on application development without worrying about the underlying infrastructure. For instance, while creating high-scaling systems, developers frequently need to write much code to manage database queries, asynchronous messaging, and caching, scaling, among other things. These features are offered by numerous PaaS solutions as a service, allowing the developers to concentrate on the business logic rather than having to code for the underlying infrastructure. NIST defines PaaS as the consumer being given the ability to deploy onto the cloud applications developed using consumer-created or purchased infrastructure libraries, programming languages, libraries, tools, and services that the cloud provider offers. The user has control over the deployed applications and may be able to configure the application-hosting environment. However, they do not manage or control the underlying cloud infrastructure, which includes the network, servers, operating systems, or storage [10].

2.2.3 Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) is the backbone of the cloud computing service model, providing a computing platform that includes virtual private networks and their network connections, virtual servers, IP addresses, load balancers, etc. In this model, the complexities associated with managing an on-premise data centre and infrastructure, such as servers, storage, and networks, are abstracted and offered as services that can be manually managed on a web-based console or even automated using the infrastructure as code approach (IaC). With IaaS, developers focus on designing and coding applications, while administrators handle the installation, management, and updates of third-party solutions. The availability of the physical infrastructure becomes a non-issue as it is taken care of by the IaaS provider [10].

2.3 Cloud providers

Cloud service providers are the core component of the cloud computing paradigm. They own the 'on-premises' hardware infrastructure that makes the software abstraction of these hardware resources possible. This section focuses on three major cloud providers - Amazon Web Services, Google Cloud Platform, and Microsoft Azure.

2.3.1 Amazon Web Services (AWS)

AWS is a subsidiary of its parent company, Amazon.com Inc. AWS provides a utility model on-demand cloud computing services to private and government-owned organisations. The genesis of AWS was the launch of the Amazon Simple Queue Service (Amazon SQS) in 2006, followed by S3 and its Elastic Compute Cloud (EC2) service.

The AWS Cloud currently spans 102 Availability Zones across 32 geographic regions worldwide, with plans to add 15 more Availability Zones and 5 more AWS Regions in Canada, Germany, Malaysia, New Zealand, and Thailand. More organisations are deciding to host their cloud-based infrastructure on AWS to benefit from improved performance, reliability, and scalability everywhere they go. AWS is rated as a Leader in the 2022 Gartner Magic Quadrant for Cloud Infrastructure and Platform Services (CIPS) for the twelfth (12th) consecutive year. Among the top eight vendors listed in the report, it is ranked highest in the context of 'Ability to Execute' as depicted in Figure 2.4. Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) are included in the CIPS magic quadrant's scope. Leaders in the quadrant are providers who set themselves apart from other hyperscalers by providing a service that can be strategically adopted and find application in a wide range of

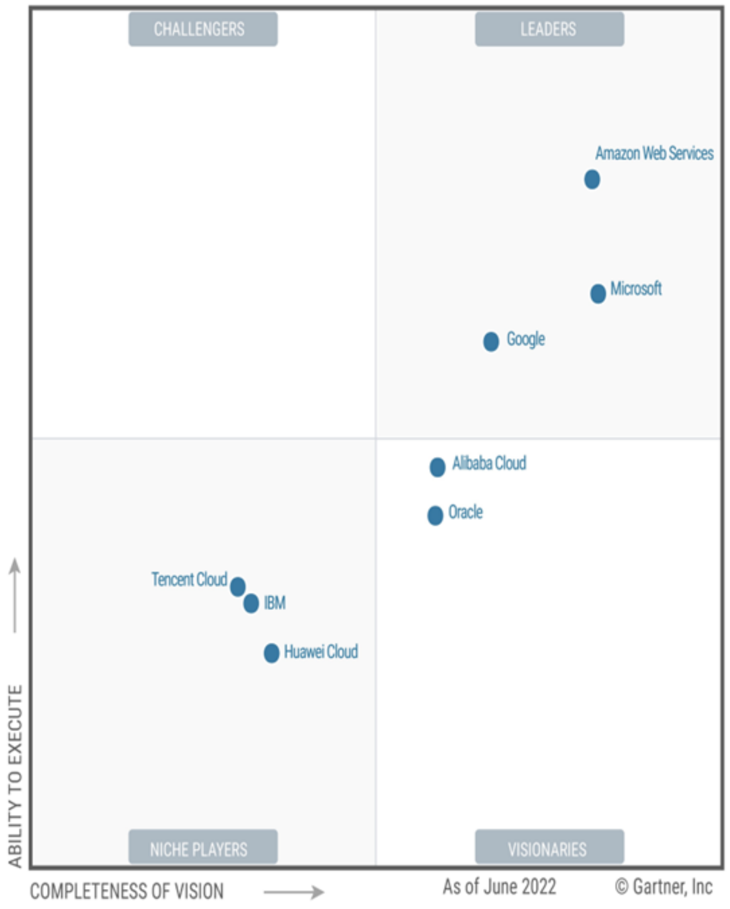


FIGURE 2.4: Gartner Magic Quadrant for Infrastructure and Platform Services 2022 [11]

use cases. Figure 2.4 above displays the quadrant, with AWS in the lead position; Google Cloud and Microsoft Azure came next. The study highlights AWS's future focus on growing to new ground with telecommunication partnerships [12].

Even while it continues to lead the cloud services industry, AWS is not without flaws. AWS has been sometimes noted to focus on short-term benefits at the price of customer relationships, according to the 2022 Gartner assessment. The restricted support that AWS provides for multi-cloud solutions is one obvious flaw in its service. An event that highlighted another worry happened on December 7, 2021, when multiple AWS services went through a protracted outage that affected users in several different parts of the world, lasting about 8 hours. Furthermore, on December 10, 2021, another 1-hour 'aftershock' service disruption occurred. These challenges, in addition to the vague communication of AWS regarding the incident, emphasise the need for significant improvements within the AWS infrastructure [13][14].

2.3.2 Microsoft Azure

Microsoft Azure is a cloud computing platform run by Microsoft. It provides cloud service models such as software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Microsoft Azure supports many programming languages, tools, and frameworks, including Microsoft-specific and third-party software and systems. Azure was first introduced at the Professional Developers Conference (PDC) in October 2008 under the "Project Red Dog."[15] It was officially launched as Windows Azure in February 2010 and later renamed Microsoft Azure on March 25, 2014 [16][17]. The first Azure offering in 2009 was SQL Azure. This was followed in 2001 by PaaS websites and IaaS virtual machines. Azure was recognised by Gartner in 2022 [13] for its robust performance in both cloud and edge computing use cases. It fits perfectly into the architecture of businesses that already have Microsoft products integrated, such as Office 365. Microsoft received recognition for its support and inclination towards hybrid and multi-cloud solutions. The majority of Azure's clients consist of medium-sized and large businesses.

Gartner CISP report for 2022 identifies specific challenges associated with Azure. Azure lacks groundbreaking innovations in IaaS and PaaS. Azure users have encountered multiple reliability and security issues. Pricing transparency was also reported as a subject of concern. Gartner's clients have reported an increase in Azure's costs over time, often without clear justification. Critical features are sometimes only available to premium-tier subscriptions, and Azure's native cost management tools lag competitors in maturity and comprehensiveness [13].

2.3.3 Google Cloud

In 2008, Google announced the launch of its Google Cloud Platform (GCP). Google's PaaS offering - Google App Engine, and its Google Storage were the first introduced offerings. This marked the entrance of Google into the cloud computing industry as a hyperscaler. Google Cloud expanded its offerings to include many services, including Google Compute Engine and Google Kubernetes Engine (GKE). Currently, Google Cloud covers 118 Zones and 187 network edge locations across 39 geographic regions. It has added new regions this year, 2023 - Berlin (Germany), Doha (Qatar), and Dammam (Saudi Arabia) with a software-defined network that provides fast and reliable connections to users worldwide [18]. Google is a Leader in the Gartner 2022 Cloud Infrastructure and Platform Service report. Google Cloud has improved its edge capabilities significantly, making it more reliable in more use cases. Google is investing more in becoming a comprehensive Infrastructure as a Service and Platform as a Service provider provider by growing the scope and magnitude of its market operations. Its customer base consists primarily of small and medium-sized businesses, and its operations are geographically diverse [13].

The strength of Google Cloud is highlighted in its revenue gain, continuous enterprise focus on selling to business executives rather than technical teams, and the willingness to embrace the idea of a sovereign cloud - it has a more flexible partnering stance outside its core regions. According to Gartner, Google Cloud outperformed all other cloud providers in the market in terms of both percentage revenue gains and enhancements across Gartner's Critical Capabilities for CIPS. However, Gartner cautioned that Google has historically attracted customers with aggressive price reductions relative to its competitors. Google has historically drawn customers using this approach, but customers should be conscious that these low prices will not last forever. For example, Google recently raised the cost of some of its storage services by 100

Chapter 3

Technical Background

The following is a list of existing tools used in this research project

3.1 Terraform

Terraform is an infrastructure as a code tool that lets you define and provision cloud and on-prem infrastructure resources as code in human-readable configuration files that you can version, reuse, and share. Terraform uses the HashiCorp Configuration Language (HCL) to define and automate infrastructure provisioning as code using a declarative approach. Terraform will be used in this project to provision the use case infrastructure to AWS Cloud [19].

3.2 Checkov

Checkov is a static code analysis tool for scanning infrastructure as code (IaC) files. The result can reveal misconfigurations that may lead to security or compliance problems. Checkov comes with about 750 predefined policies to check for common misconfiguration issues. Checkov also supports the creation and contribution of custom policies. Supported IaC types include Terraform (AWS, GCP, Azure), CloudFormation, Azure Resource Manager (ARM), Serverless framework, Helm charts, Kubernetes, and Docker. In this project, we will use Ckeckov to run a static code analysis on the code bases defined for both Terraform and AWS CloudFormation and compared the results [39].

3.3 AWS CloudFormation

AWS CloudFormation is a proprietary service that Amazon Web Services (AWS) provides. It uses YAML and JSON template files to define and automate infrastructure provisioning as code on AWS cloud environment. It deploys AWS resources in a repeatable, testable and auditable manner. It can be used to automate the provisioning of virtually every Infrastructure as a Service (IaaS) offering on the AWS Cloud. It will be used in this project to provision the use case infrastructure to AWS Cloud

3.4 Yaml

YAML is a human-readable data-serialization language. It commonly finds its use in configuration files and applications for storing or transmitting data. In this context, it will define resources in the AWS CloudFormation template files.

3.5 Git/GitHub

Git and GitHub form a dynamic combination for ´version control in software development. Git is an open-source distributed version control system. It efficiently tracks changes and manages collaborative coding efforts. GitHub is a web-based platform that is a collaborative hub for hosting and sharing Git repositories. In this project, Git and GitHub will be instrumental in employing version control and management of the IaC codes.

Chapter 4

Infrastructure as Code

Infrastructure as Code is a code-centric approach for automating infrastructure provisioning and configuration in a cloud environment, where codes and IaC tools are used to create, update, and delete cloud infrastructure resources. The idea is to define, deploy, update, and destroy your infrastructure using the codes you write and run. This connotes a significant mental shift where every facet of operations is approached as software, including those elements that correspond to hardware (e.g., physically configuring servers).

4.1 Why Infrastructure as Code?

Before the era of Cloud computing, IT systems were directly bound to physical hardware. Provisioning and maintaining infrastructures were manual, time-consuming work. Because changes involve so much work and time, change management processes emphasise careful up-front consideration, documentation, design and review [42]. The ease of provisioning new IT infrastructure and changing infrastructure configuration that came with cloud computing created a new problem where organisations had an ever-growing catalogue of systems that were difficult to manage and configurations changes that were difficult to track. Infrastructure as Code (IaC), which automates cloud infrastructure provisioning and configuration, emerged as a critical method to address this issue.

Infrastructure as Code rests on these fundamental principles [43]:

- **Automation**: Policy creation, implementation, and infrastructure management are automated through code rather than manual configuration.
- **Repeatable Proven Processes**: With IaC and version control systems, IT teams can craft proven processes and apply them automatically and repeatedly whenever the need arises.
- **Rebuild and Reproduce Systems and Configurations**: Proven processes not only apply to new systems and tasks but are used to rebuild or reconfigure existing systems that might have had a problem or need an update.

4.2 Infrastructure as Code Tools

Brikman [44] broadly categorises IaC tools into five categories:

- **Tools for Ad Hoc Scripts**: They provide a straightforward approach to automation using ad hoc scripts. Tools like Python, Bash and Ruby can define manual steps in code and the script executed on a server.
- **Configuration Management Tools**: Ansible, Chef, and Puppet are examples of configuration management tools. This means they are primarily designed to automate the installation and management of software on existing servers.
- **Server Templating Tools**: Examples of server templating tools are Docker, Packer, and Vagrant. Instead of launching multiple servers and configuring them by running the same code on each one, the idea behind server templating tools is to make use of virtualisation and containerisation technologies to create an image of a server that captures a fully self-contained "snapshot" of the operating system (OS), the software, the files, and all other relevant details [44].
- **Orchestration Tools**: While server templating tools create Virtual machines (VM) and containers, orchestration tools manage them in processes like deploying the VMs and containers, health check and monitoring, scaling up/down and distributing traffic across the VMs and containers. Examples of such tools are Kubernetes and Docker Swarm.
- **Provisioning Tools**: Provisioning tools, including Terraform, CloudFormation, and Pulumi, are in charge of building the servers in the cloud environment, while configuration management, server templating, and orchestration tools specify the code that runs on the servers. Almost every cloud infrastructure component, including databases, load balancers, security groups, subnet configurations, firewall settings and routing rules, can be created with provisioning tools.

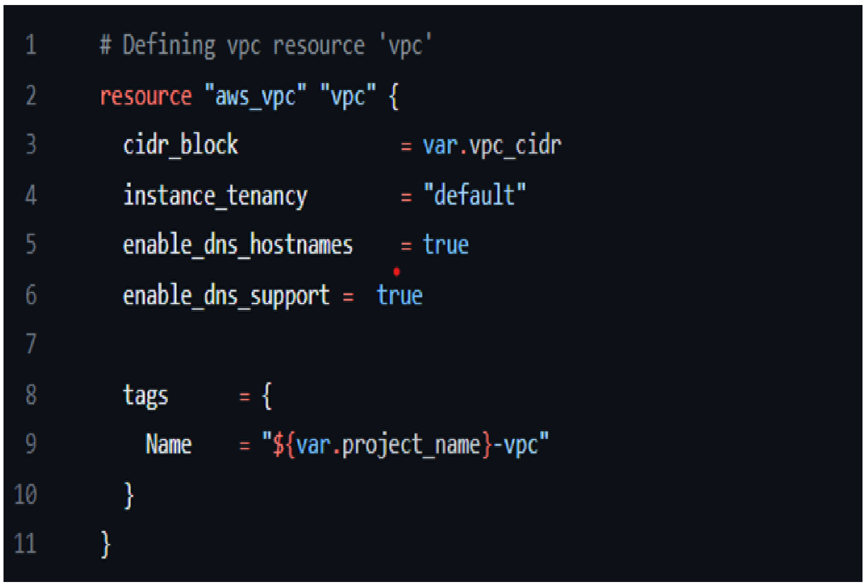
It should be noted that these categories are focused on the primary use of these tools, and the distinction is not entirely clear-cut. For example, a configuration management tool like Ansible can provision a server, and a provisioning tool like Terraform can run configuration scripts on servers. Depending on the use case, a combination might be necessary, like Terraform to provision the servers and Ansible to configure them. In this project, our focus is on two provisioning tools – AWS CloudFormation and Terraform.

4.2.1 Terraform

Terraform, developed by HashiCorp, is an open-source IaC tool designed for defining and provisioning infrastructure as code. It uses a declarative approach to enable the provisioning and management of infrastructure across private and public cloud providers. Terraform creates templates using a custom domain-specific language (DSL), or HCL. HCL, a superset of JSON, is utilised to author these templates.

An illustrative HCL file is provided in Figure 4.1, which defines the creation of a custom virtual private cloud (VPC) resource. The configuration defines the desired state of a custom VPC in a declarative approach and Terraform will create the VPC accordingly. The 'aws_vpc' resource is provided by the AWS Terraform provider. In the context of the upstream cloud, the associated Terraform providers could be AWS Google Cloud, Azure or any private cloud platform. But technically, a provider in Terraform is a logical abstraction of an upstream API. Every cloud provider has a different set of infrastructure resources: storage, databases, servers, or networks. Terraform acts as the orchestrator for controlling these resources amongst providers.

In the case of AWS, Terraform uses the AWS SDK to communicate with the AWS API layer on behalf of the AWS provider. Developers can then use Terraform HCL to specify the provider's resources, and their creation, modification, and destruction can be accomplished easily using the Terraform CLI [20].



```
# Defining vpc resource 'vpc'
resource "aws_vpc" "vpc" {
  cidr_block              = var.vpc_cidr
  instance_tenancy        = "default"
  enable_dns_hostnames    = true
  enable_dns_support =  true

  tags      = {
    Name    = "${var.project_name}-vpc"
  }
}
```

FIGURE 4.1: An example Terraform HCL file

Terraform workflow can be summarised as 'Write', 'Plan' and 'Apply'. The 'Plan' step previews the written Infrastructure as code, while the 'Apply' step provisions the reproducible infrastructure. Terraform configuration is written in any preferred code editor. Keeping your work in a version-controlled repository is standard procedure, even if you are just one person working. Running plans repeatedly as you author your configuration can help you find syntax errors and ensure everything is coming together as you had intended. This is similar to working alone on any software application code, where it is helpful to have a closed feedback loop between modifying the code and executing test commands [21].

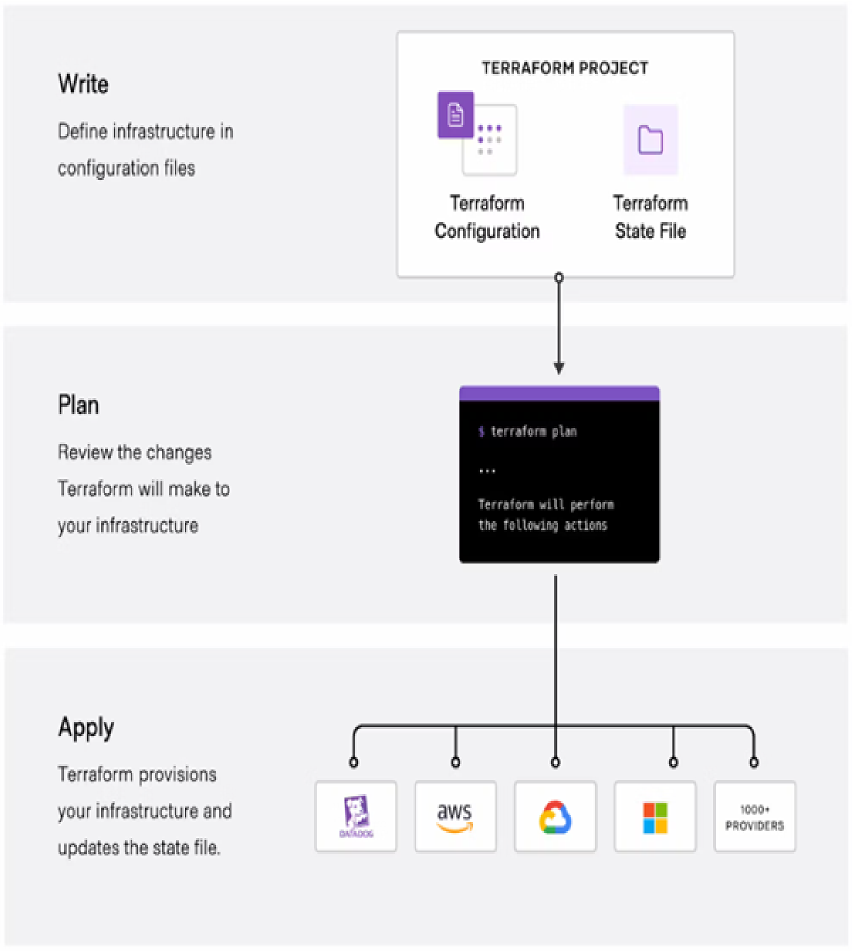

Write
Define infrastructure in configuration files
TERRAFORM PROJECT
Terraform Configuration
Terraform State File
Plan
Review the changes Terraform will make to your infrastructure
$ terraform plan
...
Terraform will perform the following actions
Apply
Terraform provisions your infrastructure and updates the state file.
aws
1000+ PROVIDERS

FIGURE 4.2: The Core Terraform Workflow [21]

4.2.2 AWS CloudFormation

AWS CloudFormation is an Infrastructure-as-code AWS service that was launched in February 2011. The goal was to streamline the modelling and configuration of AWS resources and free up more time for cloud professionals to concentrate on their AWS-running applications rather than managing them. CloudFormation handles the provisioning and configuration of the desired AWS resources (such as Custom VPC, networks, webservers (EC2), storage and database) once you create a template that lists them all. CloudFormation creates, configures, and determines which AWS resources depend on each other, saving you the trouble. The scenarios shown in Figure 4.3 indicate how AWS CloudFormation can improve efficiency [22].

AWS CloudFormation supports both Yaml and Json file format and different set of resources defined in a single template - Yaml file can be provisioned, updated or destroyed as a single CloudFormation Stack. Figure 4.4 below is an example of YAML file that defines the creation of a VPC in AWS as the cloud provider. The referenced value 'ProjectName' is a parameter already specified in a different JSON parameter file.

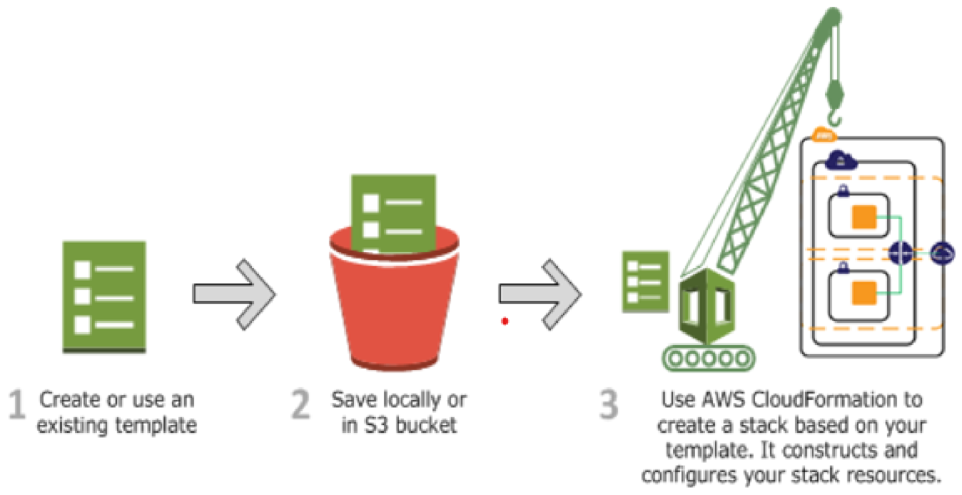
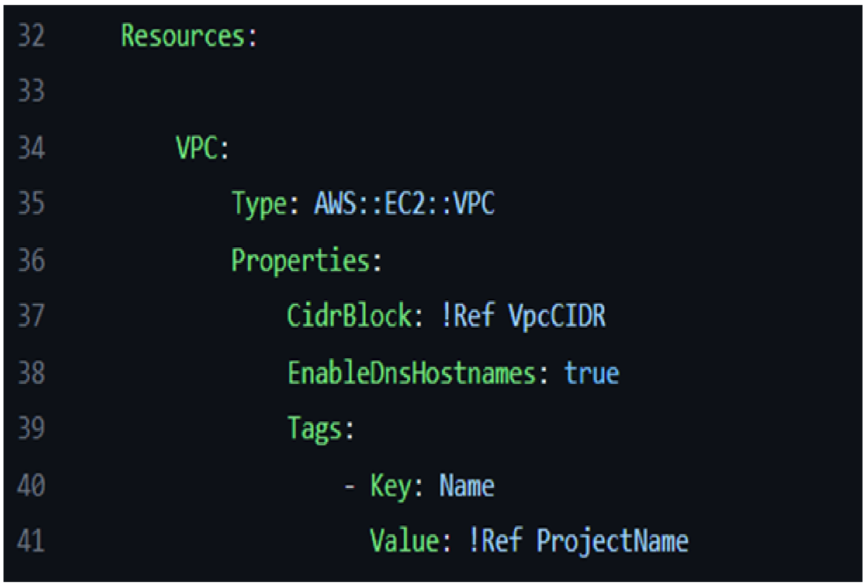

1 Create or use an existing template
2 Save locally or in S3 bucket
3 Use AWS CloudFormation to create a stack based on your template. It constructs and configures your stack resources.

FIGURE 4.3: CloudFormation workflow [23]



```
Resources:

    VPC:
        Type: AWS::EC2::VPC
        Properties:
            CidrBlock: !Ref VpcCIDR
            EnableDnsHostnames: true
            Tags:
                - Key: Name
                  Value: !Ref ProjectName
```

FIGURE 4.4: An example AWS Cloudformation YAML file

Some drawbacks of AWS CloudFormation:

- Loop-like constructs like the count meta-parameter in Terraform are impossible with AWS CloudFormation. The template logic used in AWS CloudFormation does not have the flexibility and support for such general-purpose logic.
- Delays in including support for newly released services. Hence, early adopters are tempted to rely on third-party tools to take advantage of new services quickly
- Provisioning resources in a template using the AWS CLI is more restricted as resources like 'Transform functions' to handle more complex architecture are not available in AWS Cloudformation via the CLI. And there is a threshold on the maximum size of the template and the number of resources that can be defined in a template.

Chapter 5

Comparing Terraform and AWS CloudFormation

The comparison will be based on infrastructure resource provisioning and management experience for both IaC tools on AWS Cloud. We will build and provision a specific case study infrastructure using Terraform and AWS CloudFormation.

5.1 Infrastructure Architecture

To test and compare both IaC tools, we will provision a 'High Availability WebApp Infrastructure' on AWS Cloud as a case study. The architecture is coined from the AWS WordPress reference architecture [24]. WordPress is utilised on 43.1% of all websites, and within the category of content management systems (CMS), WordPress holds a market share of about 63.0% [25].

The infrastructure architecture is designed for robust, fault-tolerant performance, incorporating several essential AWS resources to achieve scalability, resilience, and secure communication.

It includes an Internet gateway that allows communication between instances within the custom VPC and the Internet. A NAT gateway in each public subnet enables Amazon EC2 instances in the private subnets to access the internet. AWS NAT gateways serve as proxies for outbound traffic. This provides a layer of security by not exposing the EC2 instances and database instances in the private subnet directly to the public internet. The Elastic IP address in the public subnet is associated with the NAT gateway to provide a consistent IP address for outbound traffic. This is essential for scenarios where external services or servers might require a fixed IP address to whitelist for communication. An Elastic IP address is a static, public IPv4 address that does not change even if the NAT gateway needs to be replaced.

The Application Load Balancer (ALB) distributes traffic across a target group (Autoscaling Group of Amazon EC2 instances) in multiple Availability Zones (AZ-a and AZ-b). This promotes high availability and ensures efficient load distribution. With the AWS Application Load Balancers, the load balancer node that receives the request evaluates the listener rules in the defined order of priority to determine which constraint to apply, then selects a target from the target group for the rule action, using the routing algorithm configured for the target group. The default routing algorithm is round-robin [26].

A multi-availability zone (AZ) MySQL database layer in Amazon RDS is launched in private subnets to simplify database administration and ensure no direct public access. Amazon RDS is a relational database service fully managed by Amazon Cloud. It simplifies database administration tasks, such as hardware provisioning, setup, patching, and backups. The multi-AZ configuration ensures database resilience and high availability, with automatic failover to a standby replica in a different availability zone in case of any issues in the primary AZ.

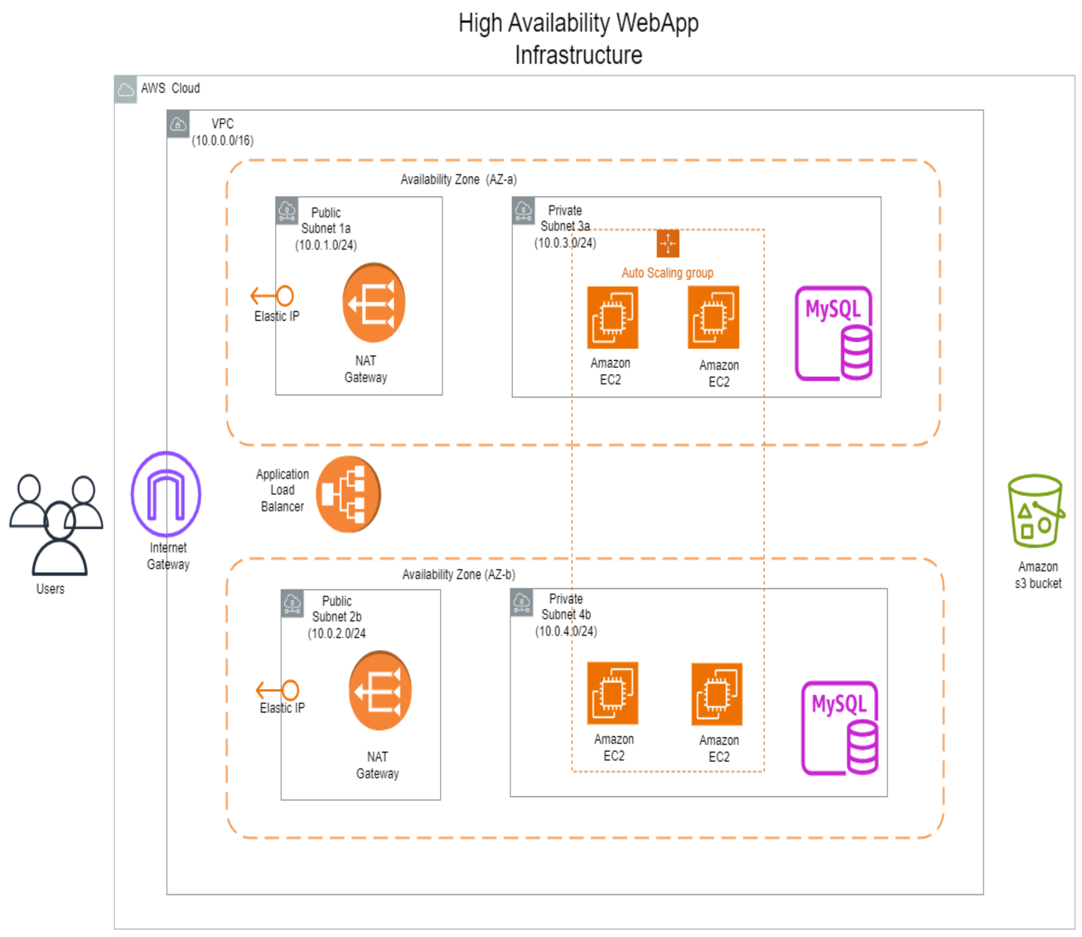
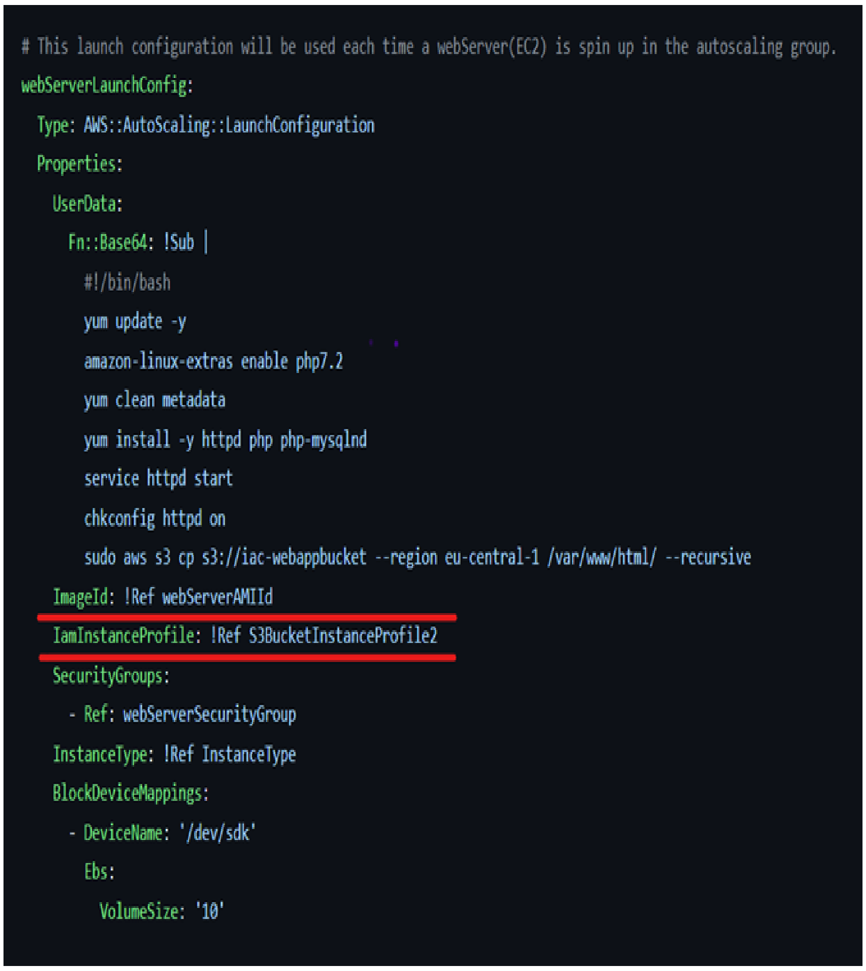


FIGURE 5.1: High Availability WebApp Infrastructure

To ensure scalability, reduced configuration management, enhanced data durability, and consistency across EC2 instances, the Amazon S3 bucket in the architecture provides a centralised and scalable storage solution. When the web server (EC2 instances) files and source codes are stored in an S3 bucket, it allows for easy scalability. As the application scales and additional EC2 instances are launched through Auto Scaling, each instance can access the same set of files from the S3 bucket seamlessly. Access to the S3 bucket is possible using an AWS Instance Profile to assign an Identity and Access Management (IAM) role to each newly launched EC2 instance in the Autoscaling Group. An AWS Instance Profile is a secure and convenient way to grant AWS IAM roles to an (EC2) instance. It allows you to associate an IAM role with an EC2 instance during its launch. In our scenario, the defined Instance Profile is provided as a parameter to the AWS launch configuration for the Autoscaling Group.



```
# This launch configuration will be used each time a webServer(EC2) is spin up in the autoscaling group.
webServerLaunchConfig:
  Type: AWS::AutoScaling::LaunchConfiguration
  Properties:
    UserData:
      Fn::Base64: !Sub |
        #!/bin/bash
        yum update -y
        amazon-linux-extras enable php7.2
        yum clean metadata
        yum install -y httpd php php-mysqlnd
        service httpd start
        chkconfig httpd on
        sudo aws s3 cp s3://iac-webappbucket --region eu-central-1 /var/www/html/ --recursive
    ImageId: !Ref webServerAMIId
    IamInstanceProfile: !Ref S3BucketInstanceProfile2
    SecurityGroups:
      - Ref: webServerSecurityGroup
    InstanceType: !Ref InstanceType
    BlockDeviceMappings:
      - DeviceName: '/dev/sdk'
        Ebs:
          VolumeSize: '10'
```

FIGURE 5.2: An example AWS CloudFormation LaunchConfiguration

An AWS LaunchConfiguration is a template that defines the settings and configuration for instances launched in an Auto Scaling group. Figure 5.2 shows the LaunchConfiguration used for provisioning this infrastructure defined for AWS CloudFormation. The IAM role has an attached policy that grants the EC2 instance access to the S3 bucket. AWS IAM policies are JSON documents that administrators use to define permissions and specific actions for AWS resources. They are used to control access to AWS services and resources.

This approach of using S3 bucket simplifies storage management and aligns with best practices for robust and scalable cloud architectures.

5.2 Implementation

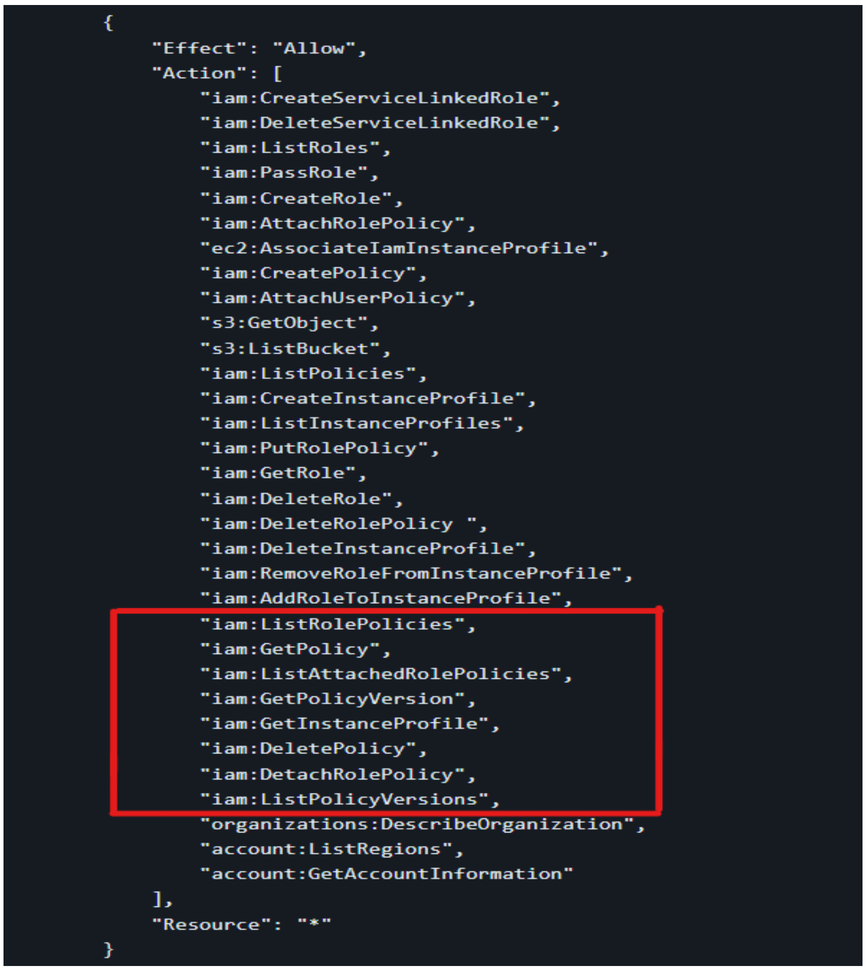
The case study infrastructure was first provisioned using AWS CloudFormatiion, then the provisioning was repeated using Terraform. The Terraform and AWS CloudFormation codes were written to provision the same case study infrastructure on AWS Cloud. The required resources were defined in each case to ensure an identical infrastructure version is provisioned on AWS Cloud. All codes written to provision this case study infrastructure on AWS Cloud are on this public GitHub [27] repository. The project is licensed under the terms of the MIT License. This licensing model permits individuals to utilise, adapt, and distribute the software freely, without restrictions.

5.2.1 Installation of required tools

Provisioning and configuring cloud resources with AWS CloudFormation and Terraform from a local Linux machine requires the installation of the AWS CLI and an AWS account with the necessary access credentials. The AWS CLI (aws-cli/2.13.16) was installed following the steps provided for Linux x86 (64-bit) on the AWS Documentation [28]. Terraform (Terraform v1.5.7) was installed using the package manager provided for Ubuntu Linux on the HashiCorp Terraform Documentation [29]. The tools were installed on a local Ubuntu 22.04.3 LTS (Intel(R) Core(TM) i7-4800MQ CPU @ 2.70GHz, 8GB RAM) Linux machine.

5.2.2 Required AWS Permissions

Figure 5.3 shows the IAM permissions required on AWS to provision the case study infrastructure for this project. It is worth noting that for the same case study infrastructure, Terraform required more AWS IAM permissions compared to AWS CloudFormation. The extra permissions required by Terraform are highlighted in Figure 5.3.



```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole",
        "iam:DeleteServiceLinkedRole",
        "iam:ListRoles",
        "iam:PassRole",
        "iam:CreateRole",
        "iam:AttachRolePolicy",
        "ec2:AssociateIamInstanceProfile",
        "iam:CreatePolicy",
        "iam:AttachUserPolicy",
        "s3:GetObject",
        "s3:ListBucket",
        "iam:ListPolicies",
        "iam:CreateInstanceProfile",
        "iam:ListInstanceProfiles",
        "iam:PutRolePolicy",
        "iam:GetRole",
        "iam:DeleteRole",
        "iam:DeleteRolePolicy ",
        "iam:DeleteInstanceProfile",
        "iam:RemoveRoleFromInstanceProfile",
        "iam:AddRoleToInstanceProfile",
        "iam:ListRolePolicies",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:GetPolicyVersion",
        "iam:GetInstanceProfile",
        "iam:DeletePolicy",
        "iam:DetachRolePolicy",
        "iam:ListPolicyVersions",
        "organizations:DescribeOrganization",
        "account:ListRegions",
        "account:GetAccountInformation"
    ],
    "Resource": "*"
}
```

FIGURE 5.3: Required AWS IAM Permissions.

5.3 Observed Performance

A performance comparison was carried out to observe each tool's provisioning time for the identical use case infrastructure on AWS Cloud. With the identical infrastructures already defined using AWS CloudFormation's JSON/YAML templates and Terraform's HCL, both tools' infrastructure creation and destruction time were observed, measured, and compared. The provisioning time for Terraform was measured using the Linux time utility tool. This command-line tool can measure the real execution time and resource utilisation of a specified command or script. It was easy to Obtain the infrastructure provisioning and destruction time for AWS CloudFormation from the time stamps on the AWS Management Console for CloudFormation resources. The experiment was repeated five (5) times for both tools to ensure statistical accuracy and reliability.

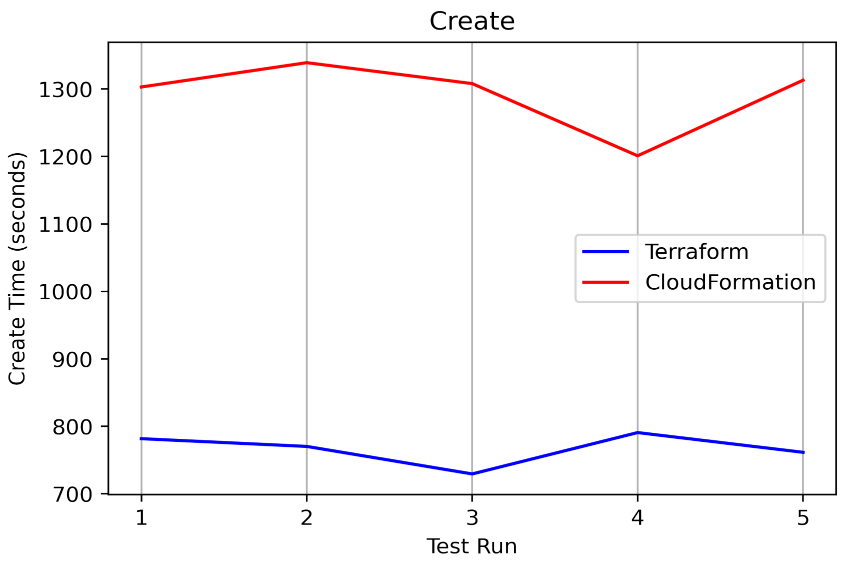


FIGURE 5.4: Infrastructure creation time

Terraform delivered faster provisioning times, as shown in Figure 5.4. The mean of the five (5) test runs showed that Terraform took an average of 767 seconds (12m, 47 sec) to provision the infrastructure. In contrast, AWS CloudFormation took an average of 1293 seconds (21min, 33sec) to provision the same case study infrastructure. The creation of the database resource took the most time for both tools.

As depicted in Figure 5.5, it was similarly observed in the destruction experiment that Terraform exhibited a notably faster destruction time with an average of 536 seconds (8m, 56 sec) for the five (5) test runs. In contrast, AWS CloudFormation recorded an average destruction time of 1019 seconds (16 minutes, 59 seconds).

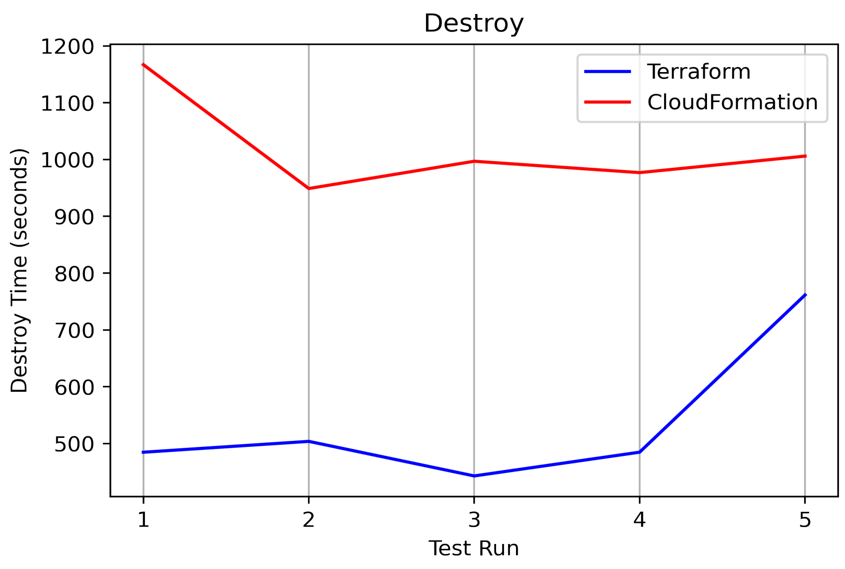


FIGURE 5.5: Infrastructure destruction time

5.4 Developer Experience

5.4.1 IDE Support

Considering Terraform (HCL, JSON) and AWS CloudFormation (YAML; JSON), we chose—visual Studio Code because of its many extensions needed for easy and efficient development. Visual Studio Code (VS Code) is a free, open-source IDE from Microsoft. It has become popular among developers due to its lightweight design and several extensions supporting different syntax and languages. We installed the HashiCorp Terraform v2.26.0 extension. It provided code validation, syntax highlighting, code formatting and code navigation.

We installed the HashiCorp Terraform v2.26.0 extension. It provided code validation, syntax highlighting, code formatting and code navigation. These functionalities made the development process easier; however, no available extension could trigger an error message for resources defined with the omission of the required property. Code errors of this kind were only discovered during the workflow's 'terraform plan' stage. Several Visual Studio Code extensions for AWS CloudFormation are available in the marketplace. We used 'aws-cloudformation-yaml v0.2.2' that adds YAML and JSON snippets to resource definitions; its features include adding the structure of output and exports to the output section, adding tags and suggesting intrinsic functions for resource definition. We also used 'aws cloud formation links v0.0.6', which can provide links to AWS Documentation for defined resources.

For both tools, it was necessary to regularly leave the IDE environment and refer back to their Documentation for the resources to understand and validate their properties. In summary, both tools had equivalent IDE support and available extensions for Visual Studio Code.

5.4.2 Learnability

Learnability was observed considering the ease of successfully executing the project task using both tools. Terraform uses HashiCorp Configuration Language (HCL) for declarative definition, while AWS CloudFormation utilises JSON and YAML templates. Our proficiency in computers, YAML and relevant Integrated Development Environment was at an expert level; however, our knowledge of HCL was at a beginner's level. Installation of Terraform was easy following the provided Documentation [29], and a user-friendly introductory tutorial [30][31] significantly contributed to a seamless onboarding experience with Terraform. AWS CloudFormation Documentation [32] stood out as regards directing users towards best practices and available templates. In the same vain, Terraform encouraged modularity; however, understanding the concept of Terraform modules and the order of resource execution proved to be less straightforward compared to the simple execution of resources in an AWS CloudFormation template.

In summary, both tools were easy to install and get started with. Still, AWS CloudFormation provided an easier learning curve towards provisioning resources and completing the project, especially with prior knowledge of the AWS Cloud environment.

5.4.3 Community support

Due to the fact that Terraform is open source and multi-cloud compared to AWS CloudFormation which is an AWS proprietary service specific to AWS, it was difficult to get accurate data that highlights the community traffic and support for AWS CloudFormation in contrast to Terraform, which has a readily available GitHub repository metric [33]. Table 5.1 highlights the present GitHub metrics for Terraform.

TABLE 5.1: Terraform GitHub Statistic

Stars	39600
Forks	9200
Contributors	1774
Pull Request	175
First Commit	2014

. The 'Star' metric serves as a measure of approval from GitHub users. It indicates the level of acceptance of an open-source project. The "Forks" metric represents new repositories that inherit code from the original Terraform repository, often necessary for idea iteration and proposing changes. 9,200 Forks, in this case for Terraform, signify a thriving community around the repository. This indicates active developer engagement and collaborative contributions.

Though AWS CloudFormation has no open-source GitHub project repository, it has a GitHub repository dedicated to its 'public coverage roadmap' with 1,100 'Stars' and 53 'Forks' [34]. It is a public roadmap is focused on upcoming coverage support for CloudFormation. It is focused on coverage additions to existing AWS services to be addressed by upcoming CloudFormation releases. However, it must be emphasized that AWS CloudFormation benefits largely from extensive AWS Documentation and support.

5.5 Key Differences in Features

5.5.1 State Management

In the context of IaC, 'state management' refers to maintaining the current state of the provisioned infrastructure with a reference to the original desired state. State management is crucial in determining if there has been any change and detecting possible drift from the defined infrastructure state.

Terraform stores its state locally in a 'terraform.tfstate' file by default. However, a remote backend is possible and recommended, especially for collaborative purposes using Terraform cloud or Amazon S3 bucket and DynamoDB. Terraform uses a 'state lock' and 'state release' mechanism to ensure that one operation is complete before the state is released for an update by another operation; this ensures that in a collaborative environment with multiple users, the problem of concurrent updates is avoided. AWS does the state management for CloudFormation by default. Users have limited visibility into the internal details of the state.

5.5.2 Modularisation

Modularisation is organising codes into reusable, self-contained modules to promote maintainability and ease of collaboration. Infrastructure architectures like a typical 'High Availability WebApp Infrastructure used in this project consist of several resources: Databases, Autoscaling Groups, Load balancers, NAT gateways, just to name a few. Defining all these resources in a single file is not recommended in the context of future maintainability; dividing the resource definition into smaller modules that are connected as needed is a better approach. Terraform employ modules to simplify IaC maintainability; it comes with native support for modules, and open-source modules are readily available in the Terraform Registry.

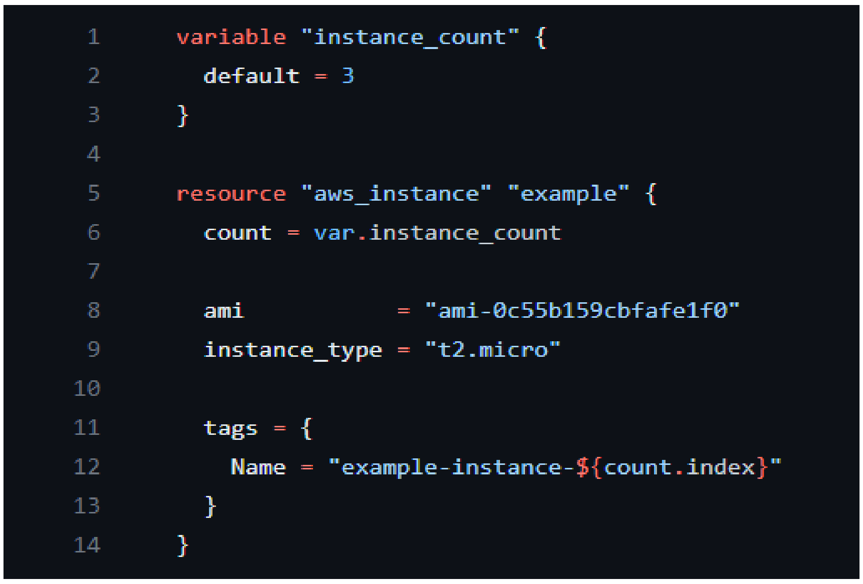
In CloudFormation, there is no native support for organising templates into modules. Though it provides features like 'Cross-Stack Reference', 'Outputs', 'Export' and 'Parameters' that the user can leverage to achieve some form of modularisation, it is up to the user to implement and manage the modularisation objective. One problem is referencing value from one stack (module) in another; the export feature solves the problem of sharing outputs, but the outputs must stay the same. Finally, CloudFormation lacks a central repository to share templates like the modules available in the Terraform registry.

5.5.3 Terraform Plan Vs Change Set

The 'Terraform Plan' command is a part of the Terraform workflow. Before creating or updating infrastructure resources, it displays a detailed execution plan describing what will be created or the changes that will be made to an existing infrastructure to achieve the desired state specified in the Terraform configuration files. This command is handy for verifying changes before a final execution. With CloudFormation, 'Change Set' is a summary of changes or updates that will be made to a CloudFormation stack. A 'Change Set' is typically associated with updates to an existing stack rather than the creation of a new stack. During the initial operation of creating a stack, there is no Change Set.

5.5.4 Loops Vs Conditions

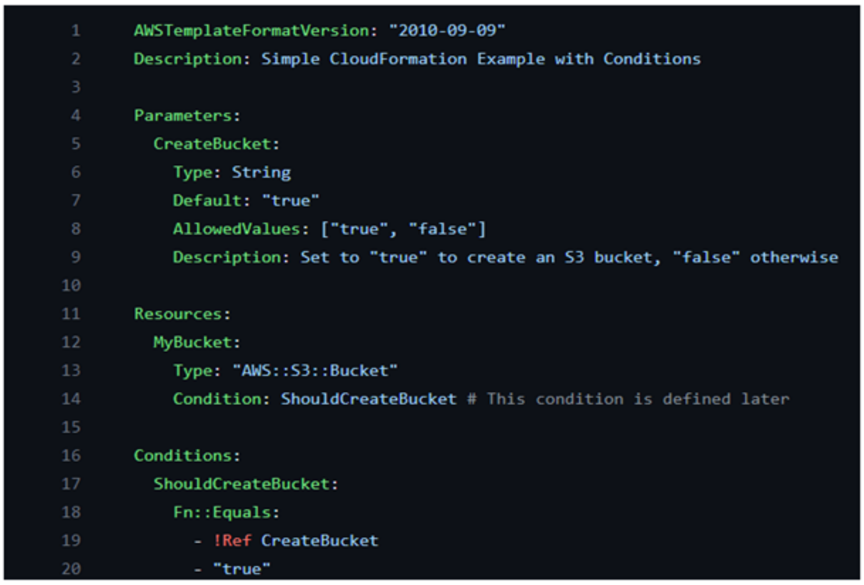
As shown in Figure 6.6 below HCL, the language used by Terraform was designed with a syntax that allows loop-like constructs like ‘for_each’ and count, which facilitate iteration through resources, enabling the dynamic creation of multiple instances, as in the example below.



```
variable "instance_count" {
  default = 3
}

resource "aws_instance" "example" {
  count = var.instance_count

  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"

  tags = {
    Name = "example-instance-${count.index}"
  }
}
```

FIGURE 5.6: Loop in Terrraform - HCL

CloudFormation uses YAML and JSON templates; these languages are focused on specifying the desired state of the resources rather than defining logic or loops. As seen in Figure 5.7, CloudFormation uses conditions, while loops are only possible in Terraform.



```
AWSTemplateFormatVersion: "2010-09-09"
Description: Simple CloudFormation Example with Conditions

Parameters:
  CreateBucket:
    Type: String
    Default: "true"
    AllowedValues: ["true", "false"]
    Description: Set to "true" to create an S3 bucket, "false" otherwise

Resources:
  MyBucket:
    Type: "AWS::S3::Bucket"
    Condition: ShouldCreateBucket # This condition is defined later

Conditions:
  ShouldCreateBucket:
    Fn::Equals:
      - !Ref CreateBucket
      - "true"
```

FIGURE 5.7: Condition in CloudFormation – YAML

5.5.5 Rollback

If a stack update fails to modify your infrastructure, CloudFormation will automatically roll back to the previous state of the infrastructure. With Terraform, there is no automatic rollback in the event of a failed provisioning; the user must manually apply the previous configuration.

5.5.6 Coverage and Scope

While CloudFormation addresses several AWS resources, it usually takes time to incorporate support for newly introduced AWS services compared to the pace of third-party tools like Terraform. Terraform also works with a wide range of AWS resources and is frequently quicker than CloudFormation in adopting new AWS resources and features. Furthermore, Terraform extends its support beyond AWS to other public and private cloud providers. At present, Terraform supports more than 3,000 integrations with more than 250 partners, including Amazon Web Services (AWS), Microsoft Azure, Google Cloud, Confluent, Datadog, MongoDB, Palo Alto Networks, ServiceNow, and Zscaler [41].

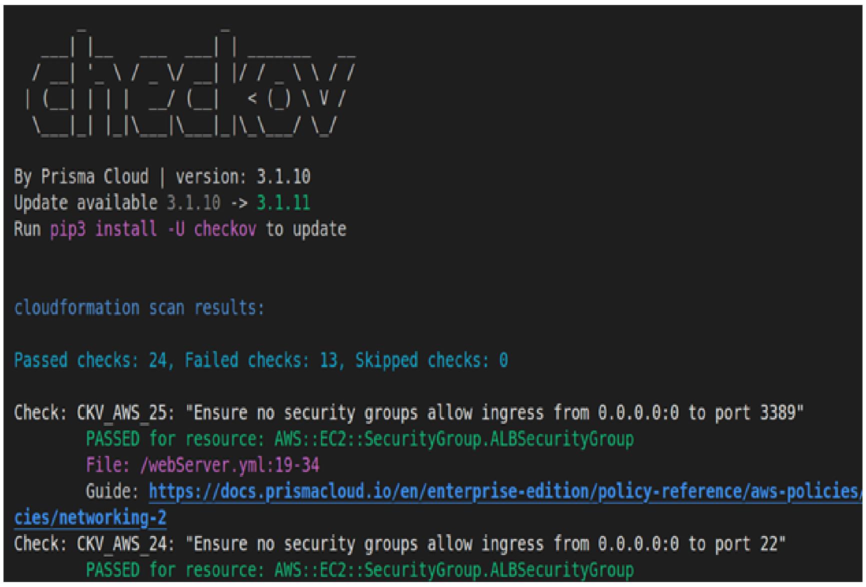
5.6 Maintainability

Maintainability is a critical consideration for Infrastructure as Code, and being specific as much as possible when naming resources and being able to segment these resources into modules that describe the architecture reduces the stress of understanding and maintaining the defined IaC, especially for large and complex infrastructures. Unlike AWS CloudFormation, where all resources in a stack are defined in one large file, Terraform uses modules to create a clearer picture of the architecture and ease maintainability and reusability. A Terraform module is a container for multiple resources that are used together. We can use modules to create lightweight abstractions to describe the infrastructure in terms of its architecture and not just in terms of the physical resources. A Terraform module is like a programming function; wherever there is repeated logic, there is an opportunity to abstract and reuse the logic. Terraform modules are the feature within Terraform that allows us to reuse infrastructure code [37][38].

5.6.1 Code Analysis with Checkov

There are different static code analysis tools for IaC. However, to further compare the maintainability of Infrastructure as Code defined for Terraform and CloudFormation, we used a tool, Checkov, that could run static code analysis on IaC files defined using Terraform or CloudFormation. The result can reveal misconfigurations that may lead to security or compliance problems. Checkov comes with about 750 predefined policies to check for common misconfiguration issues. Checkov also supports the creation and contribution of custom policies. Supported IaC types include Terraform (AWS, GCP, Azure), CloudFormation, Azure Resource Manager (ARM), Serverless framework, Helm charts, Kubernetes, and Docker [39].

We used Python's package manager, pip, to install the latest version of Checkov (3.1.10) with the latest version of Python in our system. The instructions provided in the Checkov documentation [39] for running the tests on both code bases were easy to follow. When Checkov scans a directory containing CloudFormation templates, it will validate if the resource definitions in the files comply with the best practices defined by AWS. Checkov adheres to the style of CloudFormation template reference, wherein specific resources may incorporate an attribute reference leading to the ultimate state or scenarios of possible dependency between resources.



```
       _               _
   ___| |__   ___  ___| | _______   __
  / __| '_ \ / _ \/ __| |/ / _ \ \ / /
 | (__| | | |  __/ (__|   < (_) \ V /
  \___|_| |_|\___|\___|_|\_\___/ \_/

By Prisma Cloud | version: 3.1.10
Update available 3.1.10 -> 3.1.11
Run pip3 install -U checkov to update


cloudformation scan results:

Passed checks: 24, Failed checks: 13, Skipped checks: 0

Check: CKV_AWS_25: "Ensure no security groups allow ingress from 0.0.0.0:0 to port 3389"
        PASSED for resource: AWS::EC2::SecurityGroup.ALBSecurityGroup
        File: /webServer.yml:19-34
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/
cies/networking-2
Check: CKV_AWS_24: "Ensure no security groups allow ingress from 0.0.0.0:0 to port 22"
        PASSED for resource: AWS::EC2::SecurityGroup.ALBSecurityGroup
```

FIGURE 5.8: An Example Checkov Scan Result for CloudFormation

For Terraform, resources defined in .tf files can be evaluated using Checkov. Additionally, Checkov can scan Terraform plans expressed in the JSON file format; this gives Checkov access to additional dependencies and more precise context to provide better scan results. Since Terraform plan files may contain arguments (like secrets) that are injected dynamically, it is usually considered best as part of best practice to run a plan evaluation using Checkov in a secure CI/CD pipeline setting [40].

TABLE 5.2: Checkov Scan Result

Checkov Checks	**Terraform**	**CloudFormation**
Passed Checks	45	24
Failed Checks	30	13
Skipped Checks	0	0
Total Checks	75	37
Pass rate	60%	65%
Fail rate	40%	35%

Table 5.2 above displays the Checkov scan result for this project code base. Seventy-five (75) checks ran for the .tf files contained in the Terraform folder, with 30 failed checks and a pass rate of 60%, while 24 checks ran for the CloudFormation resource files, with 13 failed checks and a pass rate of 65%. Though some of the failed checks could be ignored, for the development of secure Infrastructure as Code, it is crucial to have corresponding tools like Checkov to assist in validating and correcting the configuration of the defined resources in line with the best practices outlined by the cloud provider.

Chapter 6

Conclusion

This research compared two infrastructures as code tools, Terraform and AWS Cloud-Formation, for provisioning and configuring cloud resources on Amazon Web Services. The comparative study assessed each tool's performance, key features, and advantages, focusing on their suitability for infrastructure resource provisioning and configuration on AWS.

An example case study infrastructure was selected for comparison based on its generalised application. The comparison involved creating the example infrastructure as code and provisioning it on AWS using both IaC tools. For provisioning the same case study infrastructure on AWS, we noticed Terraform required more AWS IAM permissions than AWS CloudFormation.

However, in the performance evaluation, Terraform demonstrated superior speed in infrastructure provisioning and destruction operations. However, AWS Cloud-Formation stood out as more user-friendly as regards the learning curve. Terraform provided an enhanced developer experience, especially with regard to its modularity as the infrastructure development became more complex. Terraform's provision for the use of modules was invaluable in providing a clear picture of the defined infrastructure and ease of maintainability, especially for large and complex infrastructures. The procedure for static code analysis as a measure of maintainability was similar for both tools; we used an existing software tool, 'Checkov', to run a static code analysis on both Terraform and AWS CloudFormation code bases.

AWS CloudFormation is recommended for beginners familiar with the AWS cloud infrastructure environment. Its simple YAML and JSON approach provides a more readable and easy-to-implement template. AWS CloudFormation's guidance in development through the extensive Documentation provided by AWS lowers the entry barrier, making the initial development phase quick and straightforward. Perhaps a business logic for why organisations should use CloudFormation on AWS Environment is that it is under the AWS support offerings as an AWS service. Hence, receiving support for bug fixes for deployments on the AWS cloud would be easier than using a third-party tool like Terraform

For more experienced users who have a clear vision of the required infrastructure in the respective cloud environment, Terraform is a stable and mature tool that allows declarative programming using a domain-specific language. It excels in scenarios where deployment and update times are critical. Terraform's flexibility in working with multiple cloud providers makes it suitable for large software projects with evolving cloud requirements, avoiding vendor lock-in to AWS.

Despite the evaluation, certain limitations should be acknowledged, including the focus on the AWS ecosystem, specific criteria and scenarios, and the limited use of AWS services and infrastructure complexity. It must be emphasised that though the result of the experiment in this study provides valuable insight into the performance difference between Terraform and AWS CloudFormation, it is essential to note that the focus was on a specific case study that cannot fit into all scenarios. And not all AWS resources were considered.

Future work should explore performance, portability, scalability characteristics, and integrating both tools with other DevOps practices and tools. Additionally, user surveys or case studies could provide valuable insights into practical implications and challenges in real-world environments, aiding in decision-making for adopting these tools based on specific project requirements and team preferences. Ultimately, the choice between Terraform and AWS CloudFormation should align with project complexity, familiarity with the domain-specific language and cloud environment, AWS service integration needs, and multi-cloud support requirements. Each tool has its strengths and limitations, and the decision should be tailored to the unique needs of the development team.

References

[1] (Yang, Xiaoyu, et al 2014). "Cloud Computing in E-science: Research Challenges and Opportunities." [Online] https://doi.org/10.1007/s11227-014-1251-5

[2] (Gartner, 2022) Gartner Magic Quadrant for Cloud Infrastructure and Platform Services 2022 [Online] Available on: https://aws.amazon.com/blogs/aws/aws-named-as-a-leader-in-the-2022-gartner-cloud-infrastructure-platform-services-cips-magic-quadrant-for-the-12th-consecutive-year/

[3] (Flexera, 2022) Flexera State of the Cloud Report 2022 [Online] Available on: https://path.flexera.com/cm/report-state-of-the-

[4] (Coccia M. et al. 2018) Comparative Studies DOI: 10.1007/978-3-319-31816-5_1197-1 Available on:
https://www.researchgate.net/publication/323573952_Comparative_Studies

[5] (Mell P, Grance T, 2011) The NIST definition of cloud computing [Online] DOI: 10.6028/NIST.SP.800-145 Available on:
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[6] (Fang Liu, Jin Tong et al. 2011) “NIST Cloud computing reference Architecture”, NIST Special Publication 500-292, July 2011

[7] (Nawsher Khan et al, 2012), “Cloud Computing Architecture for Efficient Provision of Services”, Network-Based Information System (NBiS), 2012 15th International Conference on, September 2012

[8] (Adam Hayes, 2023) Dotcom Bubble Definition [Online]
Available on: https://www.investopedia.com/terms/d/dotcom-bubble.asp#

[9] (SlidNetwork-Basedeteam, 2023). Cloud service models - comparison between SaaS, PaaS and IaaS models [Online]
Available on:https://www.slideteam.net/cloud-service-models-it-comparison-between-iaas-paas-and-saas-models.html

[10] (Michael Kavis, 2014). Architecting the cloud: Design Decisions for Cloud Computing Models [E-book] Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

[11] (Gartner, 2022) Gartner Magic Quadrant for Infrastructure and Platform Services 2022 [Online] Available on:
https://aws.amazon.com/blogs/aws/aws-named-as-a-leader-in-the-2022-gartner-cloud-infrastructure-platform-services-cips-magic-quadrant-for-the-12th-consecutive-year/

[12] AWS Global Infrastructure, 2023). [Online] Available on:
https://aws.amazon.com/about-aws/global-infrastructure/

[13] (Raj Bala et al, 2022) Gartner Magic Quadrant for Infrastructure and Platform Services 2022 [Online] Available on:
https://services.google.com/fh/files/misc/gartner_magic_quadrant_cips_2022.pdf

[14] (ThousandEyes, 2021) AWS Outage Analysis: December 7, 2021 [Online] Available on: https://www.thousandeyes.com/blog/aws-outage-analysis-dec-7-2021

[15] (Wikipedia, 2023) Microsoft Azure [Online] Available on:
https://en.wikipedia.org/wiki/Microsoft_Azure

[16] (Abandy, Roosevelt 2022). "The History of Microsoft Azure". Microsoft Tech Community. [Online] Available on:
https://techcommunity.microsoft.com/t5/educator-developer-blog/the-history-of-microsoft-azure/ba-p/3574204

[17] (Microsoft Azure. 2014) "Upcoming Name Change for Windows Azure" [Online] Available on:
https://web.archive.org/web/20180726184658/https://azure.microsoft.com/
Archived from the original on:
July 26, 2018. Retrieved August 29, 2016.

[18] (Google Cloud 2023) Google Locations [Online]
Available on: https://cloud.google.com/about/locations

[19] (HashiCorp Terraform Tutorial, 2023) What is Terraform [Online] Available on:
https://developer.hashicorp.com/terraform/intro

[20] (Bradley, 2020) Craft Infrastructure-as-Code Solutions [Ebook] DOI: 10.1007/978-1-4842-5398-4

[21] (Terraform Tutorial, 2023) The Core Terraform Workflow [Online] Available on:
https://developer.hashicorp.com/terraform/intro/core-workflow

[22] (AWS Documentation, 2023) What is AWS CloudFormation? [Online]
Available on:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

[23] (AWS Documentation, 2023) AWS CloudFormation [Online] Available on:
https://aws.amazon.com/cloudformation/

[24] (AWS samples, 2023) aws-refarch-wordpress [Online] Available on:
https://github.com/aws-samples/aws-refarch-wordpress

[25] (W3Techs, 2023) Usage statistics of content management systems [Online] Available on: https://w3techs.com/technologies/overview/content_management

[26] (AWS Documentation, 2023) How Elastic Load Balancing works [Online] Available on: How Elastic Load Balancing works

[27] (Igbinehi, 2023) https://github.com/matthewesosa/IaC-Research_Project

[28] (AWS Documentation, 2023) Install or update the latest version of the AWS CLI [Online] Available on: https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

[29] (HarshiCorp Terraform Documentation, 2023) Install Terraform [Online] Available on: https://developer.hashicorp.com/terraform/install

[30] (Brown, 2023) Terraform Cloud Project Beginner Bootcamp [Online] Available on: https://app.exampro.co/student/material/terraform-cpb/5373

[31] (HarshiCorp Terraform Tutorial, 2023) Build infrastructure [Online] Available on: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-build

[32] (AWS Documentation, 2023) AWS CloudFormation best practices [Online] Available on: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html

[33] (HashiCorp, 2023) Terraform GitHub Repository [Online] Available on: https://github.com/hashicorp/terraform

[34] (AWS, 2023) CloudFormation Public Coverage Roadmap [Online] Available on: https://github.com/aws-cloudformation/cloudformation-coverage-roadmap#introduction

[35] (KodeKloud, 2023) Terraform vs. CloudFormation: A Side-by-Side Comparison [Online] Available on: https://kodekloud.com/blog/terraform-vs-cloudformation/

[36] (Modern Technologist, 2023) Comparing Terraform and CloudFormation [Online] Available on: https://moderntechnologist.com/terraform-vs-cloudformation/

[37] (HarshiCorp Terraform Documentation, 2023) Creating Modules [Online] Available on: https://developer.hashicorp.com/terraform/language/modules/develop

[38] (Lou Bichard, 2019) Terraform Modules: A Guide to Maintainable Infrastructure as Code [Online] Available on: https://openupthecloud.com/terraform-modules-tutorial/#google_vignette

[39] (Checkov Documentation, 2023) What is Checkov [Online] Available on: https://www.checkov.io/1.Welcome/What%20is%20Checkov.html

[40] (Checkov, 2023) Terraform Plan Scanning [Online] Available on: https://www.checkov.io/7.Scan

[41] (Melar Chen, 2023) HashiCorp Terraform Integrations [Online] Available on: https://www.hashicorp.com/blog/hashicorp-terraform-ecosystem-passes-3-000-providers-with-over-250-partners

[42] (Morris, 2021) Infrastructure as Code: Managing Servers in the Cloud [Ebook] Available on: https://books.google.de

[43] (Chef, 2023) What is Infrastructure as Code (IaC)? [Online] Available on: https://www.chef.io/glossary/what-is-infrastructure-as-code?

[44] (Brikman, 2022) Terraform: Up & Running [Ebook] isbn=9781098116743 O'Reilly Media, Inc