INFRASTRUCTURE AS CODE TOOLS COMPARISON ON AWS CLOUD ENVIRONMENT

SONGWUT COTCHARAT

A Thematic Paper Submitted in Partial Fulfillment of the
Requirements for the Degree of Master of Engineering
Department of Computer Engineering,
College of Innovative Technology and Engineering
Dhurakij Pundit University
Academic Year 2022

ใบรับรองสารนิพนธ์

วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์ มหาวิทยาลัยธุรกิจบัณฑิตย์

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต หลักสูตรวิศวกรรมคอมพิวเตอร์

หัวข้อสารนิพนธ์	INFRASTRUCTURE AS CODE TOOLS COMPARISON ON AWS CLOUD ENVIRONMENT
เสนอโดย	นายทรงวุฒิ คชรัตน์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาสารนิพนธ์	ดร.ชัยพร เขมะภาตะพันธ์

ได้พิจารณาเห็นชอบโดยคณะกรรมการสอบสารนิพนธ์แล้ว

[illegible]......................................ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.ณรงค์เดช กีรติพรานนท์)

[illegible]...................................... กรรมการและอาจารย์ที่ปรึกษาสารนิพนธ์

(อาจารย์ ดร.ชัยพร เขมะภาตะพันธ์)

[illegible]......................................กรรมการ

(อาจารย์ ดร.ธนัญ จารุวิทยโกวิท)

วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์รับรองแล้ว

[illegible]..คณบดีวิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์

(อาจารย์ ดร.ชัยพร เขมะภาตะพันธ์)

วันที่..๒๒...เดือน...มิถุนายน......พ.ศ..๒๕๖๖......

Thematic Paper Title	INFRASTRUCTURE AS CODE TOOLS COMPARISON ON AWS CLOUD ENVIRONMENT
Author	Songwut Cotcharat
Thematic Paper Advisor	Dr. Chaiyaporn Khemapatapan
Program	Computer Engineering
Academic Year	2022

ABSTRACT

In the agile world, characterized by an increased demand for speed and IT projects, teams are increasingly reliant on task automation. Consequently, Infrastructure engineers face the challenge of effectively managing their workloads. When it comes to cloud-based infrastructure, Infrastructure as Code (IaC) serves as an effective means of automating manual tasks, offering benefits such as scalability, speed, and transparency. Given its recent emergence, there are numerous IaC tools available for selection. The most commonly tools used in the industry are AWS CloudFormation and Terraform. This study aims to compare these tools and determine which one is more suitable. The research methodology involved conducting a survey to build a three-tier application infrastructure on AWS using IaC, while simultaneously examining and comparing the results obtained from these tools. The findings indicate that, in terms of superiority, there is no significant disparity between Terraform and CloudFormation; however, both tools may require substantial investments of time and resources due to their inherent complexity. The choice between these tools also depends on the specific requirements and preferences associated with building applications or infrastructures. Moreover, it is worth noting that the IaC community is rapidly expanding, and comprehensive support for advanced use cases can be easily found in official documentation. In summary, effectively evaluating and comparing these tools proves to be a challenging task. Nevertheless, this research provides valuable insights into their functionalities and how they can be effectively employed within various environments.

[illegible]

Advisor

ACKNOWLEDGEMENT

I would like to express my sincere thanks to my advisor, Dr. Chaiyaporn Khemapatapan for his valuable help and constant patient and encouragement throughout the course of this research. I am most grateful for his teaching and advice, not only the research methodologies but also many other methodologies in life. I would not have achieved this research without all the support that I have always received from him. Finally, I most gratefully acknowledge all of my classmate for all their support throughout the period of the study and research.

Songwut Cotcharat

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

CHAPTER 1
INTRODUCTION

1.1 Problem and Background

In the modern day, the ability to provision, configuration, and deployment of the infrastructure and application are very challenging and very competitive. The engineer must have a various knowledge in almost every technology field. The old ways to do this is that everything needs to be done manually and individually. The problem starts with on how you plan to buy a hardware, how you can configure all the essential tools to be compile with your application, and how you deploy your applications. This process could take months to complete before you can start a real work on that project. Not to include how difficult you have to maintenance and monitoring your own infrastructure. A lot of engineering time will spend on those topics.

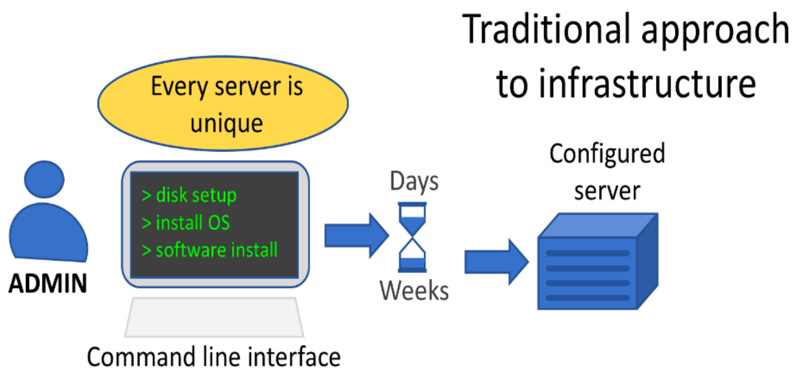


Figure 1.1 Traditional Setup Infrastructure

Source: Bloom, D. (2017)

The Organizations that can provision, configuration their infrastructure, and deploy the application early and with high frequency will have the ability to compete in the market. A new approach called DevOps and Infrastructure as a code (IaC) promise to allow the organization to reach these goals. Many organizations seem interested in this new approach of organizing development and operations, as shown by the number of publications dealing with DevOps in popular press. DevOps has also become a topic of active scientific research, as demonstrated by the increasing number of scientific papers published on the topic. Infrastructure as a code (IaC) is the practice to automatically configured system dependencies and to provision local and remote instances. Practitioners consider IaC as a fundamental pillar to implement DevOps practices, which helps them to rapidly deliver software and services to end-users. Information technology (IT) organizations, such as Facebook, Google, and Netflix have adopted IaC.

If you work in a technical team that builds and runs IT infrastructure, then cloud and infrastructure automation technology should help you deliver more value in less time, and to do it more reliably. But in practice, it drives ever-increasing size, complexity, and diversity of things to manage. These technologies are especially relevant as organizations become digital. “Digital” is how people in business attire say that software systems are essential to what the organization does. The move to digital increases the pressure on you to do more and to do it faster. You need to add and support more services. More business activities. More employees. More customers, suppliers, and other stakeholders. Cloud and automation tools help by making it far easier to add and change infrastructure. But many teams struggle to find enough time to keep up with the infrastructure they already have. Making it easier to create even more stuff to manage is unhelpful.

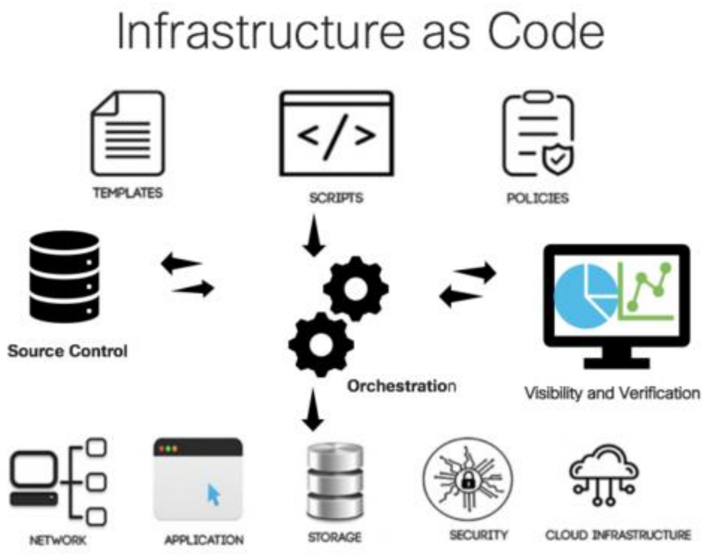


Figure 1.2 Basic Infrastructure as Code

Source: Taylor, T. (2022)

1.2 Purpose/Objective

1.2.1 To measure the efficiency and the effectiveness of two IaC tools to use on AWS (Terraform and AWS CloudFormation).

1.2.2 To help individual or organization decide which tools they should select for their works.

1.2.3 To bring introduction to the new technology for personal or business use.

1.2.4 To prove that the traditional ways of how-to setup infrastructure and application should not be the best anymore.

1.3 Research Scope

1.3.1 Design a 3-tier web application that needs to be built on Amazon Web Service(AWS).

1.3.2 Write a terraform and AWS cloud formation script to build a same infrastructure on Amazon Web Service (AWS).

1.3.3 Start build the infrastructure on the centralize personal host machine.

1.3.4 Analyze the result by comparing these topics.

1.3.4.1 The capabilities of both tools (What it can and cannot do).

1.3.4.2 The efficiency of both tools (Measuring time consuming, resource consu-ming).

1.3.4.3 Error Handling

1.3.4.4 Rollback ability

1.4 Tools

1.4.1 Personal computer with 10 Cores CPU and 16 GB of Memory

1.4.2 MAC OS

1.4.3 AWS CloudFormation

1.4.4 Terraform Software

1.4.5 AWS CLI

1.4.6 AWS Account

1.4.7 VS Code Text Editor

1.5 Research Action Plan

Table 1.1 Research Action Plan

Action	TimeLine (Months)											
	1	2	3	4	5	6	7	8	9	10	11	12
1. Research and Collect Information	🟨	🟨	🟨									
2. Infrastructure Design on AWS Cloud			🟨	🟨								
3. Study Terraform Programming tool				🟨	🟨							
4. Study AWS Cloud Formation tools					🟨	🟨						
5. Create a terraform script						🟨	🟨					
6. Create an AWS Cloud Formation script								🟨	🟨			
7. build the infrastructure on AWS using both tools										🟨		
8. Analyze and adjust according to the result											🟨	
9. Summarize the conclusion and make the report												🟨

1.6 Expected Benefit

1.6.1 Introduce an infrastructure as code tool and concept to wider audience.

1.6.2 Can help business in term of cost optimization for using infrastructure as code instead of the traditional way of provisioning.

1.6.3 Can help business improve daily productivity for an engineer by using more faster and reliable infrastructure as code tools.

CHAPTER 2

CONCEPT, THEORY AND LITERATURE REVIEW

2.1 Related Concept and Theory

2.1.1 Cloud Computing

Cloud computing is a relatively recent and currently very high-profile method of IT Deployment. Compute facilities such as virtualized server hosting or remote storage accessed via an API, are provided by a cloud provider to a cloud client. Often a client is a third party company, who will use the compute facilities to provide an external service to their users. For example, Amazon is a cloud provider supplying a storage API to Dropbox, who use the API to provide a file synchronization service to domestic and commercial users. In many ways, cloud computing resembles a move away from conventional desktop computing that has been the hallmark of the previous decade, towards a centralized computing model. One stark difference to the mainframes of the timesharing past is that the new paradigm is supported by vast data centers containing tens of thousands of servers working in unison. These datacenters are described as 'Warehouse-Sized Computers' (WSCs), reflecting their view that the machines can collectively be regarded as a single entity (Artac et al., 2017). The coordination of these machines is supported by specialist software layers, including virtualization technologies.

Most of the technologies involved in constructing and using cloud computing are not new, but rather it is their particular combination and large-scale deployment that is novel. The emergence of cloud computing would not have been possible without the growth in virtualization and widespread internet access. Cloud computing promises to commoditize data storage, processing and serving in the way envisioned by utility computing and service oriented architecture.

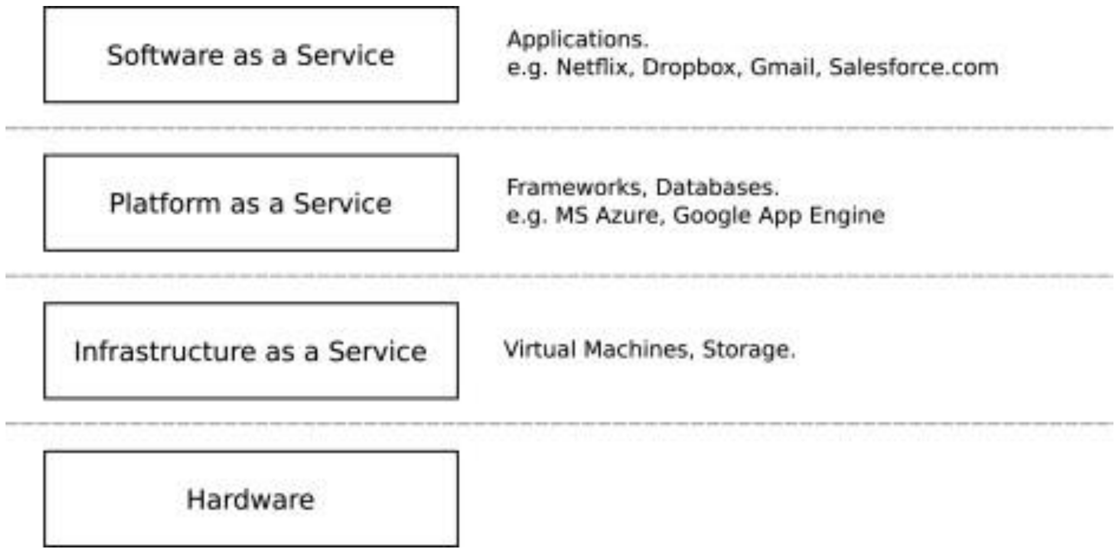

Software as a Service
Applications.
e.g. Netflix, Dropbox, Gmail, Salesforce.com
Platform as a Service
Frameworks, Databases.
e.g. MS Azure, Google App Engine
Infrastructure as a Service
Virtual Machines, Storage.
Hardware

Figure 2.1 The Cloud stack architecture

Cloud computing is frequently presented as a layered architecture, as shown in However, in practice these layers are not distinct. For example, data storage services may be regarded as infrastructure, a platform or an application, depending on exactly how those services are being used. We deliberately avoid using these somewhat artificial distinctions.

Cloud hardware is typically composed of commodity server-grade ×86 computers arranged in a shallow hierarchy composed of racks and clusters of racks. They are physically located in a series of geographically distributed data centers. In the case of Amazon Web Services, these data centers are split into discrete availability zones. Servers run hypervisor technology such as Xen or VMWare, which manage multiple virtual machines (VMs) on each physical machine. VMs are provided directly to client companies by providers such as Rackspace or Amazon at the infrastructure layer. Each live virtual machine is an *instance* of an offered VM configuration, which is principally defined by the memory and processing power available to the VM. Every instance mounts a *machine image*, also known as an *operating system image*. An image includes an operating system, and the required server software such as web servers or transaction processing software.

The most straightforward example of a platform-level service is Google App Engine, where a customer provides code and Google automatically manages the scaling to respond to incoming requests. Applications comprise most usage of cloud computing by the general public. Popular current examples include Microsoft Office 365 and Gmail.

2.1.2 DevOps

The word "DevOps" is a mashup of "development' and "operations" (Alexander, S. 2020) but it represents a set of ideas and practices much larger than those two terms alone, or together. DevOps includes security, collaborative ways of working, data analytics, and many other things. DevOps describes approaches to speeding up the processes by which an idea goes from development to deployment in a production environment where it can provide value to the user. These approaches require that development teams and operations teams communicate frequently and approach their work with empathy for their teammates. Scalability and flexible provisioning are also necessary. With DevOps, those that need power the most, get it through self service and automation. Developers, usually coding in a standard development environment, work closely with IT operations to speed software builds, tests, and releases without sacrificing reliability.

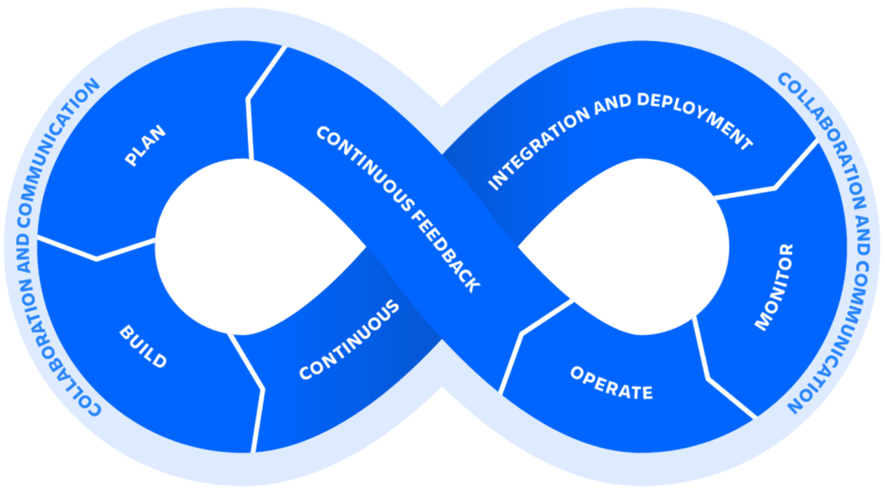

COLLABORATION AND COMMUNICATION
PLAN
BUILD
CONTINUOUS
CONTINUOUS FEEDBACK
INTEGRATION AND DEPLOYMENT
MONITOR
OPERATE
COLLABORATION AND COMMUNICATION

Figure 2.2 DevOps lifecycle

Source: Ramamoorthy, C. (2021)

Because of the continuous nature of DevOps, practitioners use the infinity loop to show how the phases of the DevOps lifecycle relate to each other. Despite appearing to flow sequentially, the loop symbolizes the need for constant collaboration and iterative improvement throughout the entire lifecycle (Artac et al., 2017).

2.1.3 Automation Provisioning

Provisioning is the process of creating and setting up IT infrastructure, and includes the steps required to manage user and system access to various resources. Provisioning is an early stage in the deployment of servers, applications, network components, storage, edge devices, and more.

Provisioning is not the same thing as configuration management, but they are both steps in the deployment process. Once a system has been provisioned, the next step is to configure the system and maintain it consistently over time.

Server provisioning is the process of setting up physical or virtual hardware; installing and configuring software, such as the operating system and applications; and connecting it to middleware, network, and storage components. Provisioning can encompass all of the operations needed to create a new machine and bring it to the desired state, which is defined according to business requirements.

Cloud provisioning includes creating the underlying infrastructure for cloud environment, like installing networking elements, services, and more. Once the basic cloud infrastructure is in place, provisioning involves setting up the resources, services, and applications inside a cloud.

User provisioning is a type of identity management that involves granting permissions to services and applications within a corporate environment like email, a database, or a network often based on a user's job title or area of responsibility. The act of revoking user access is often referred to as deprovisioning.

An example of user provisioning is role based access control (RBAC). Configuring RBAC includes assigning user accounts to a group, defining the group's role for example, read-only, editor, or administrator and then granting these roles access rights to specific resources based on the users' functional needs.

When referring to IT infrastructure, network provisioning is the setting up of components such as routers, switches, and firewalls; allocating IP addresses; and performing operational health checks and fact gathering. For telecommunications companies, the term "network provisioning" refers to providing users with a telecommunications service, such as assigning a phone number, or installing equipment and wiring.

Service provisioning includes the set up of IT-dependent services for an end user and managing the related data. Examples of service provisioning may include granting an employee access to a software-as-a-service platform, and setting up credentials and system privileges to limit access to certain types of data and activities.

Provisioning often requires IT teams to repeat the same process over and over, such as granting a developer access to a virtual machine in order to deploy and test a new application. This makes manually provisioning resources time-consuming and prone to human error, which can delay the time-to-market of new products and services. Manual provisioning also pulls busy IT teams away from projects that are more important to an organization's larger strategy. Today, most provisioning tasks can easily be handled through automation, using infrastructure-as-code (IaC). With IaC, infrastructure specifications are stored in configuration files, which means that developers just need to execute a script to provision the same environment every time. Codifying infrastructure gives IT teams a template to follow for provisioning, and although this can still be accomplished manually, automation tools can make this process far more efficient

Using repeatable workflows, automated provisioning provides greater consistency across modern IT environments, reduces the likelihood of errors and loss of productivity, and frees IT teams to focus on strategic business objectives. With this more efficient provisioning process:

- End users and developers can gain access to the IT resources and systems they need in less time, empowering them to be more productive.
- Developers can bring applications and services to market faster, which can improve customer experience and revenue.
- IT teams can spend less time on menial, repetitive tasks like correcting errors and misconfigurations, which allows them to focus on more critical priorities.

2.1.4 Infrastructure as Code (IaC)

Infrastructure as Code uses DevOps methodology and versioning with a descriptive model to define and deploy infrastructure, such as networks, virtual machines, load balancers, and connection topologies (Brikman, Y. 2019). Just as the same source code always generates the same binary, an IaC model generates the same environment every time it deploys.

The value of adopting IaC may not be apparent to stakeholders especially the ones that do not work with infrastructure. The practice of IaC is often thought of as chaotic and unnecessary. There is no need in automating something that is going to be built only once. However, even the best IT systems tend to get updates or become broken. The complex and time consuming is example how cloud resources are deployed with manual processes. A developer by means of UI or CLI sporadically creates, updates, or deletes resources. This process could happen a multitude of times by more than one developer. As a result, rarely anyone in the team would have a clear understanding about the state of the resources (Guerriero et al., 2019).

In the long-term, IaC makes it easier to add repeatability to the infrastructure and understand future changes as every change follows the same deployment process. On Figure 2 an example flow chart shows how IaC could be utilized, is presented. The infrastructure gets documented, tested, and deployed automatically and it will always be pushed to version control and have a visible history of deployment history. When infrastructure code is ready, the resources can be re-created again with a click of a button in case of failure, update, or breaking change. Defining infrastructure in an automated manner removes the need for routine tasks and save developers' time to focus on improving development systems.

Adoption of IaC is less likely to happened in old fashioned or monolithic oriented teams. In these teams, automation is not a concern, and they tend to think that infrastructure should be handled by someone else. IaC need arises in teams with certain software development practices. One of the main characteristic factors is the use of cloud computing to host applications and infrastructure. As resources' numbers starts to grow, it becomes very difficult to keep track of the state of the environment. IaC can prove itself useful and time saving to manage those resources. Teams might also be applying Agile or DevOps and find IaC to be the way to make infrastructure provisioning easier. When starting with IaC teams should be aware about version control and code review practices as both are important for knowing about the changes that are happening to the environment. To provide a full automation capability with IaC it also should be integrated with CI/CD pipelines. IaC aids developers to continuously learn and improve systems as developers learn to be afraid of breaking the infrastructure.

There is no best scenario for how to use IaC tools within a project as every project is unique. The IaC community is still in a process of learning and discovering better practices. In each case IaC should be tailored to a project's needs to fulfil its potential. IaC can be applied in projects in multiple ways: from moving a part of existing infrastructure to the cloud, to building an application based on a microservice architecture and other cloud offerings. Developers might as well start small and only deploy single resources in an automated way.

It can also be beneficial to have the infrastructure code grow together with the application code.

Supporting a multi-environment and multi-tenant setup is another case where IaC provides most benefits. Instead of spending days to recreate the same environment manually, adding a new customer can be done by adding a new parameters file and setting up an additional deployment task. A test environment can also be created and destroyed on demand to reduce costs of running resources that are used only occasionally.

Starting to write infrastructure in a form of code can be challenging especially if previously GUI was used to perform these tasks. There is a long list of decisions that developers should make prior to setting up IaC. They need to decide which tool to pick from a wide array of different tools. IaC code resides in version control and developers need to plan the location and structure of their code. Every environment needs to be configured to properly store parameters and secrets. Before the code is executed it should be tested and after it is running be updated and follow upkeep rules. As developers create infrastructure, they can decide to define some resources as modules that can be used to speed up creation of future resources also in subsequent projects.

For consulting companies that operate by projects and create resources into their customer's environment, IaC can bring both advantages and problems. On the one side it allows repeatability and consistency for the infrastructure, conversely it generates difficulties related to permissions and compliment with customers' security policies. Some customers are not convinced by automated resource provisioning as it can generate additional costs. Often there is no professional on their side who is experienced enough to approve the code. For some customers it might also get difficult to explain the overhead cost that the initial IaC work requires

Starting to write infrastructure in a form of code can be challenging especially if previously GUI was used to perform these tasks. There is a long list of decisions that developers should make prior to setting up IaC. They need to decide which tool to pick from a wide array of different tools. IaC code resides in version control and developers need to plan the location and structure of their code. Every environment needs to be configured to properly store parameters and secrets. Before the code is executed it should be tested and after it is running be updated and follow upkeep rules. As developers create infrastructure, they can decide to define some resources as modules that can be used to speed up creation

of future resources also in subsequent projects. For consulting companies that operate by projects and create resources into their customer's environment, IaC can bring both advantages and problems. On the one side it allows repeatability and consistency for the infrastructure, conversely it generates difficulties related to permissions and compliment with customers' security policies. Some customers are not convinced by automated resource provisioning as it can generate additional costs. Often there is no professional on their side who is experienced enough to approve the code. For some customers it might also get difficult to explain the overhead cost that the initial IaC work requires.

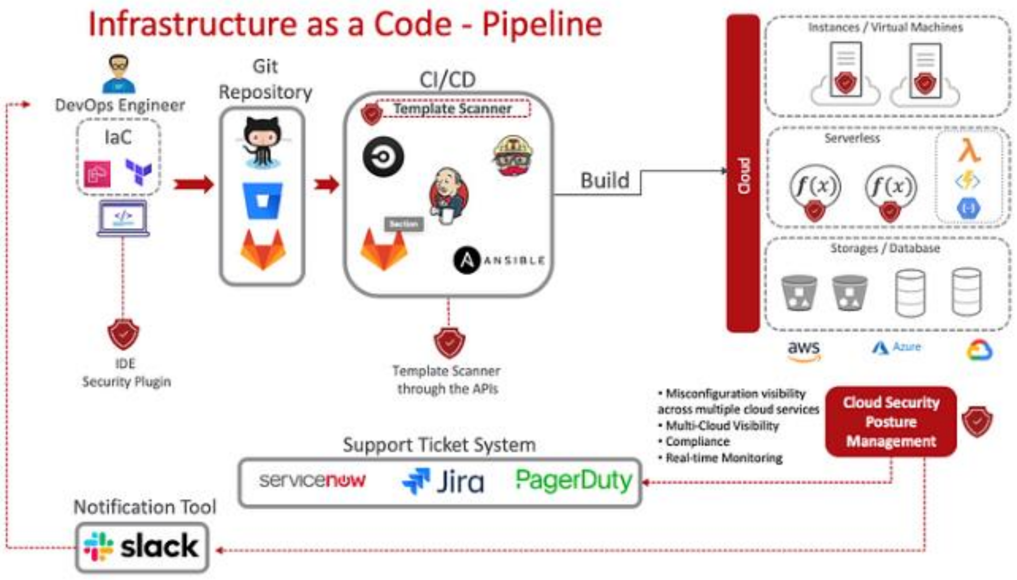


Figure 2.3 Infrastructure as a Code-Pipeline Source

Source: Mike (2022)

2.1.5 Infrastructure as Code on Cloud

A fundamental principle of DevOps is to treat infrastructure the same way developers treat code. Application code has a defined format and syntax. If the code is not written according to the rules of the programming language, applications cannot be created. Code is stored in a version management or source control system that logs a history of code development, changes, and bug fixes. When code is compiled or built into applications, we expect a consistent application to be created, and the build is repeatable and reliable (Labouardy, M. 2021).

Practicing infrastructure as code means applying the same rigor of application code development to infrastructure provisioning. All configurations should be defined in a declarative way and stored in a source control system. Infrastructure provisioning, orchestration, and deployment should also support the use of the infrastructure as code.

Infrastructure was traditionally provisioned using a combination of scripts and manual processes. Sometimes these scripts were stored in version control systems or documented step by step in text files or run-books. Often the person writing the run books is not the same person executing these scripts or following through the run-books. If these scripts or runbooks are not updated frequently, they can potentially become a show-stopper in deployments. This results in the creation of new environments not always being repeatable, reliable, or consistent.

In contrast, AWS provides a DevOps-focused way of creating and maintaining infrastructure. Similar to the way software developers write application code, AWS provides services that enable the creation, deployment and maintenance of infrastructure in a programmatic, descriptive, and declarative way. These services provide rigor, clarity, and reliability. The AWS services discussed in this paper are core to a DevOps methodology and form the underpinnings of numerous higher-level AWS DevOps principles and practices (Campbell, B. 2020).

2.1.5.1 AWS CloudFormation, One of the most used IaC on Cloud tools is AWS CloudFormation.It is a service provided by Amazon Web Services that enables users to model and manage infrastructure resources in an automated and secure manner. Using CloudFormation, developers and engineers can define and provision AWS infrastructure resources using a JSON or YAML formatted Infrastructure as Code template (Contino, 2022).

2.1.5.2 Terraform, another one of the most famous tools is Terraform. Terraform manages external resources (such as public cloud infrastructure, private cloud infrastructure, network appliances, software as a service, and platform as a service) with "providers" (Alexander, S. 2020). Hashi Corp maintains an extensive list of official providers and can also integrate with community-developed providers. Users can interact with Terraform providers by declaring resources or by calling data sources. Rather than using imperative commands to provision resources, Terraform uses declarative configuration to describe the desired final state. Once a user invokes Terraform on a given resource, Terraform will perform create, read, update and delete actions on the user's behalf to accomplish the desired state. The infrastructure as code can be written as modules, promoting reusability and maintainability. Terraform plugin protocol is built gRPC, that allow terraform to communicate to the target provider (zero Fruit, 2022).

2.1.5.3 Terraform and CloudFormation Physical Differences, the primary difference between Terraform and CloudFormation is that Terraform is a multi-cloud platform, while CloudFormation is specific to AWS. Terraform provides a common language to define and provision cloud infrastructure, while CloudFormation is an AWS-specific solution that provides a standard way to provision and manage AWS resources. Please see a side-by-side comparison of Terraform and CloudFormation below.

Table 2.1 Terraform and CloudFormation Physical Differences

Criteria	Terraform	CloudFormation
Language	HCL (Terraform own language)	JSON or YMAL
State Management	User Managed	AWS Managed
AWS Compatibility	Yes	Yes
Multi Cloud	Yes	No
Cost	Free	Free

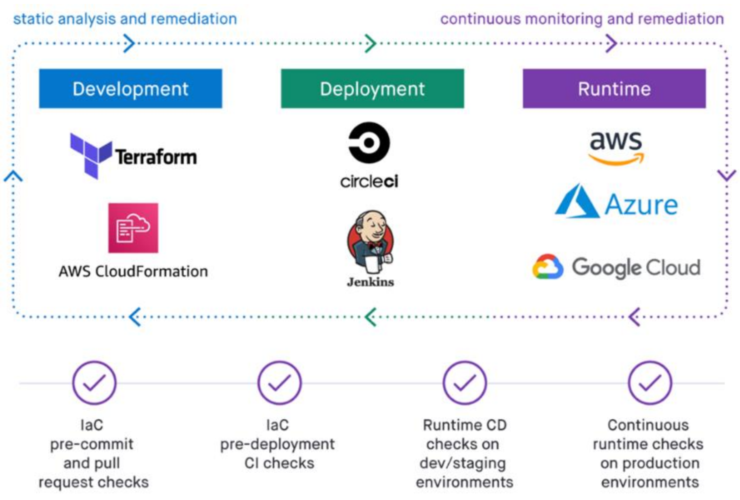

static analysis and remediation
continuous monitoring and remediation
Development
Deployment
Runtime
Terraform
AWS CloudFormation
circleci
Jenkins
aws
Azure
Google Cloud
IaC pre-commit and pull request checks
IaC pre-deployment CI checks
Runtime CD checks on dev/staging environments
Continuous runtime checks on production environments

Figure 2.4 Using IaC on Cloud

Source: Josh, S. (2021)

2.2 Literature Review

2.2.1 Adoption of Infrastructure as Code (IaC) in Real World

Olga Murphy, JAMK University of Applied Sciences did a research an adaptation of how the real world of infrastructure as a code. The conclusion was manual processes require more time and are error prone, but IaC provides developers with the ability to automate creation of infrastructure and support Cloud Data Ecosystems in an automated manner. Adoption of IaC has a potential to lower companies' expenses while improving time efficiency, consistency, and transparency of their cloud infrastructure. As mentioned by Gartner and Hashi Corp, there is clearly a lack of professionals that are competent enough to automate more complex tasks and environments. Without these professionals, successful implementation of IaC is greatly reduced and the potential benefits to industry would be negligible. To get more IaC competent and fluent developers, industry wide adoption will be necessary. A big part of adoption will include successfully learning IaC and opening discussion for its development. From both the literature review and the results of the survey the best strategies for learning IaC are to partition the subject into smaller, easily absorbed pieces and convey full teams to follow said practices. However, this learning will require time and repetition, along with commitment from management, customers and developers in companies.

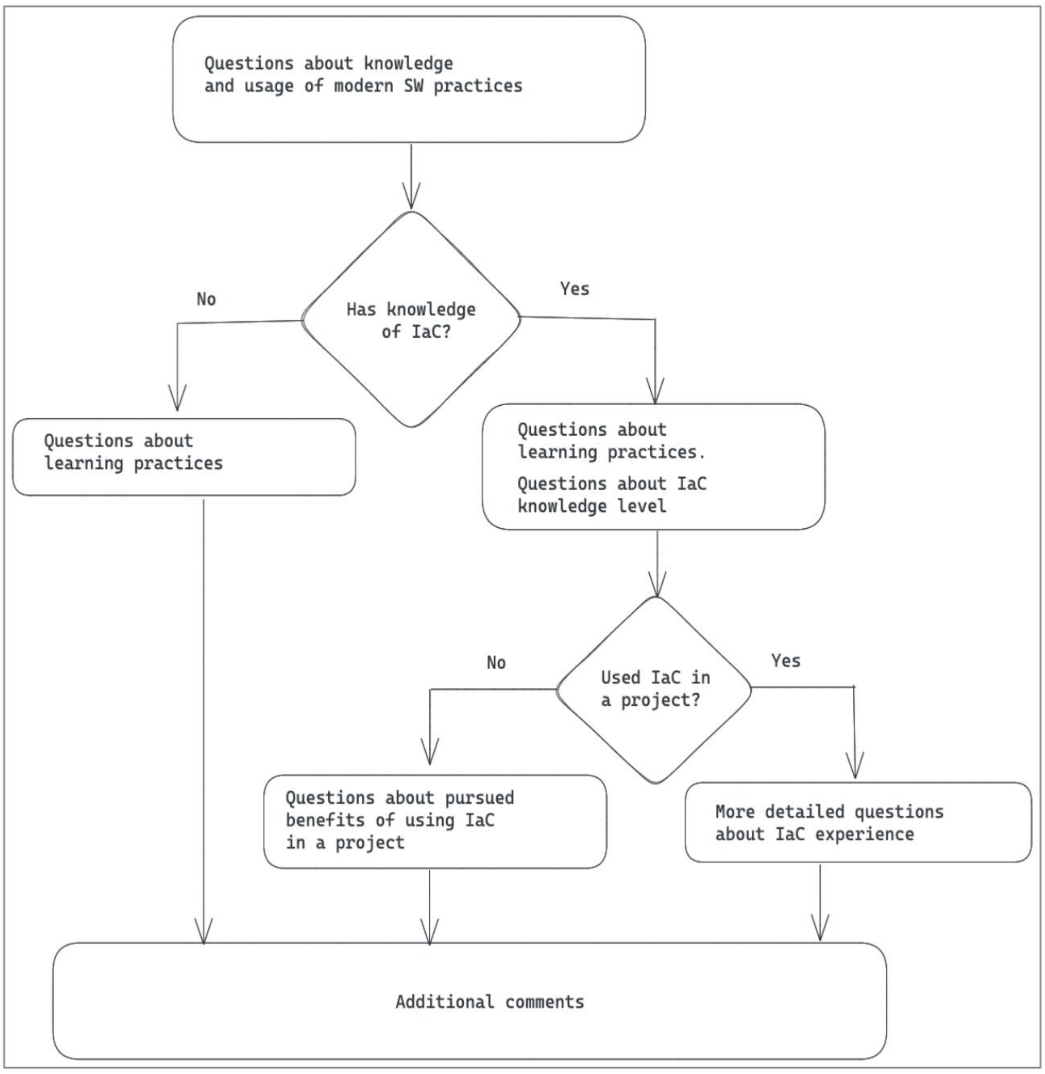

Questions about knowledge and usage of modern SW practices
Has knowledge of IaC?
No
Yes
Questions about learning practices
Questions about learning practices. Questions about IaC knowledge level
Used IaC in a project?
No
Yes
Questions about pursued benefits of using IaC in a project
More detailed questions about IaC experience
Additional comments

Figure 2.5 Example Research Workflow

Source: Olga Murphy, (2021)

IaC generates significant interest from both academia and industry. There has been an increasing number of publications and blogs on the subject in recent years. However, more structured research is required to develop best practices and elaborate on the usage of tools. Furthermore, more in-depth case studies and advanced knowledge is needed to broaden insights and expertise in the field. One possible research avenue could be analyzing learning materials provided by IaC tools’ vendors to see how successfully they convey core IaC practices and how coherent their examples are to the needs of the field. It is through investment of time and resources into research and learning that Infrastructure as Code would become economically relevant. Once it generates enough interest from the customers it can become truly widespread and reach its final adoption stage.

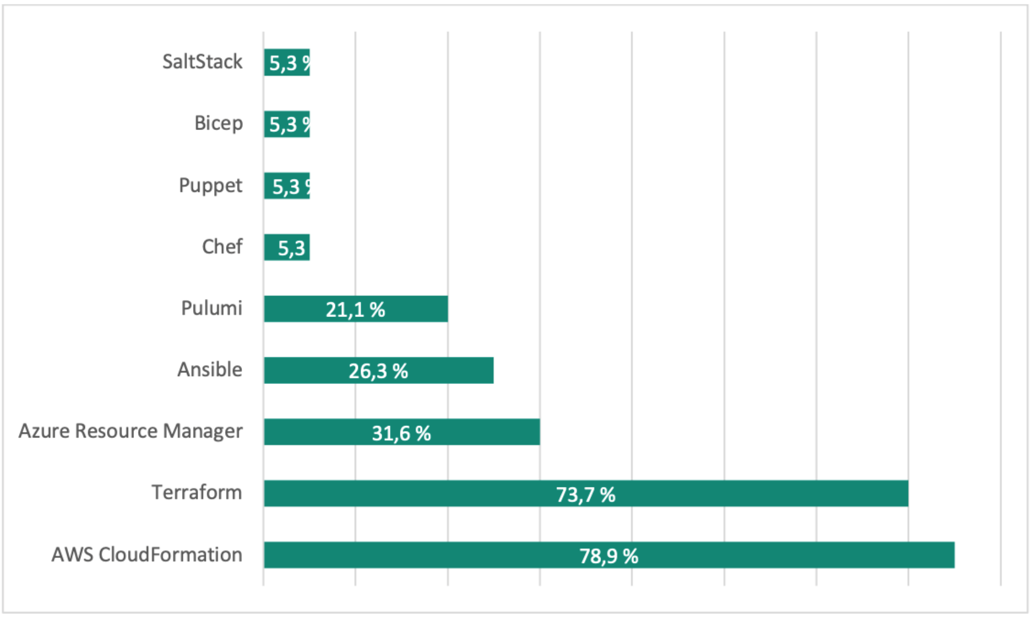


Figure 2.6 IaC Tools usages

Source: Olga Murphy, (2021)

2.2.2 Systematic mapping study of IaC

Rahman A. at 41st International Conference on Software Engineering completed a systematic mapping study of IaC and concluded that IaC as a trend is growing but is still under-researched. They identified gaps in the existing literature connected to IaC. The authors recommend multiple directions for the researchers that are investigating the IaC subject. They emphasise a lack of empirical studies, defect analysis, security, anti-patterns, and knowledge and training. IaC supports DevOps processes by simplifying resource provisioning. While DevOps is mostly about organisational culture, IaC is focused on the tools and how those tools are used by practitioners within teams and projects. As various tools are emerging quickly it takes efforts to learn them. While there is a vast number of sources and grey literature available, there is no de facto standard on how IaC should be implemented. This is due to multiple reasons: tools are diverse, use cases for IaC are broad and often are not well defined.

2.2.3 Challenges of adopting agile methods in a public organization

International Journal of Information Systems and Project Management Nuottila, J., Aaltonen, K., & Kujala, J. conducted a case study of DevOps in multiple companies. They have observed the DevOps practices and collected feedback on how each case identified the goals and benefits of DevOps adoption. They also identified IaC practices separately from DevOps context. The authors discussed the benefits of IaC while noting that in some cases scripts have been provisioned by the operations teams. The same cases described that developers did not have much knowledge of IaC but that it could be acquired in the future from the operations team.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Methodology

3.1.1 Study, research, analyze on how we should provision the server on public cloud-(AWS) by using infrastructure as code called “Terraform” and “CloudFormation” and compare both tools efficiency. To get the result we wanted on what should be the recommendation tools for organization or company to use to get the better experience and reduce the workload on infrastructure provisioning. Include study about an emerging technology that can be adapted to use to accomplish the goal and the result for this research.

3.1.2 Analyze and Design a 3-tier web application that needs to be built on Amazon Web Service (AWS).

3.1.3 Build and write a terraform and AWS cloud formation script to build a same infra-structure on Amazon Web Service (AWS).

3.1.4 Start building the infrastructure on the centralize personal host machine by running them step by step.

3.1.5 Test running all the environments by using both tools on the same infrastructure environments.

3.1.6 Compare both tools and analyze the test result according to the methodology that has been designed and Improve the result.

3.1.7 Conclusion and produce the report.

3.2 Research Tools

3.2.1 Personal computer with 10 Cores CPU and 16 GB of Memory

3.2.2 MAC OS

3.2.3 AWS CloudFormation

3.2.4 Terraform Software

3.2.5 AWS CLI

3.2.6 AWS Account

3.2.7 VS Code Text Editor

3.3 System Development and Implementation

The process will start by using IaC tools building AWS infrastructure 3 tier web application with high availability and high scalability. The 3-tier web application will include VPC, Internet Gateway, Security group, Application Load Balancer, Private and Public Subnets, EC2 with Auto scaling group, and Database with high availability.

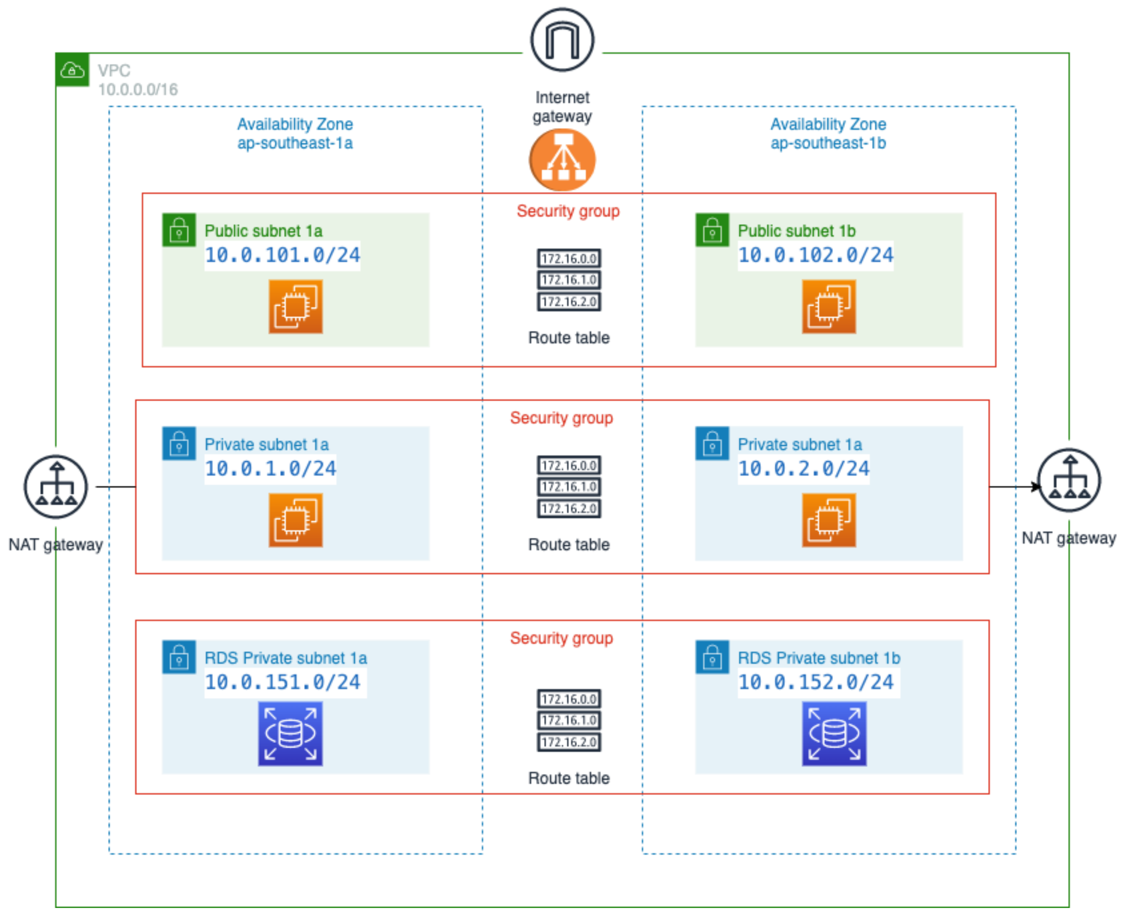


Figure 3.1 Overall System Design

From Figure 3.1, We will create 3 tier web application that include web tier which will access from the internet. Application tier, which will access by only private subnet, and lastly DB tier which again can access to only a private subnet. All 3 tier will form a web application infrastructure provisioning that will be ready to use by a developer. All of this will using terraform and CloudFormation stacks to provision them.

3.3.1 Terraform Implementation, in this step terraform script will be created to build a-3tier web application which consist with all the infrastructure module script in it.

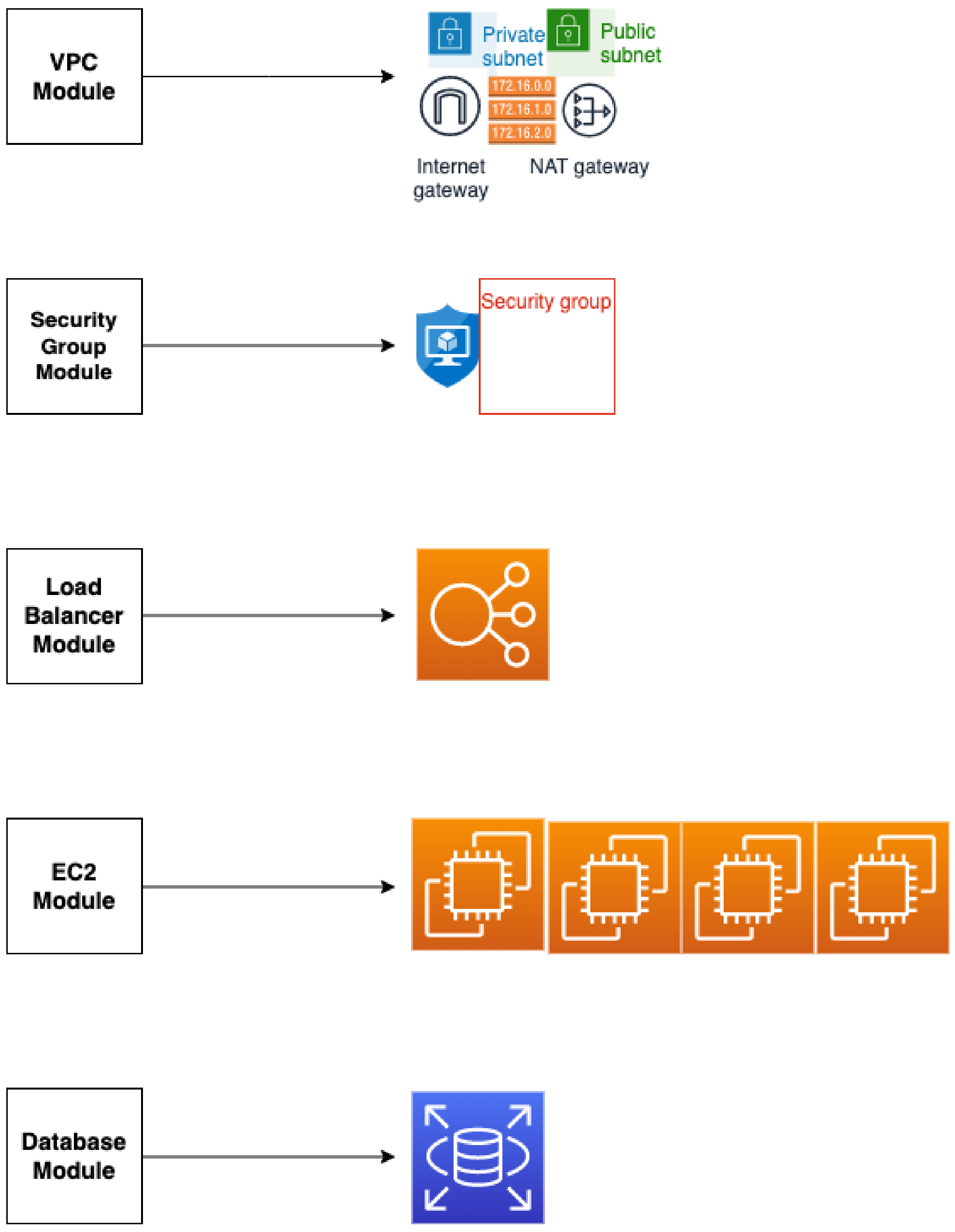


Figure 3.2 Terraform Design

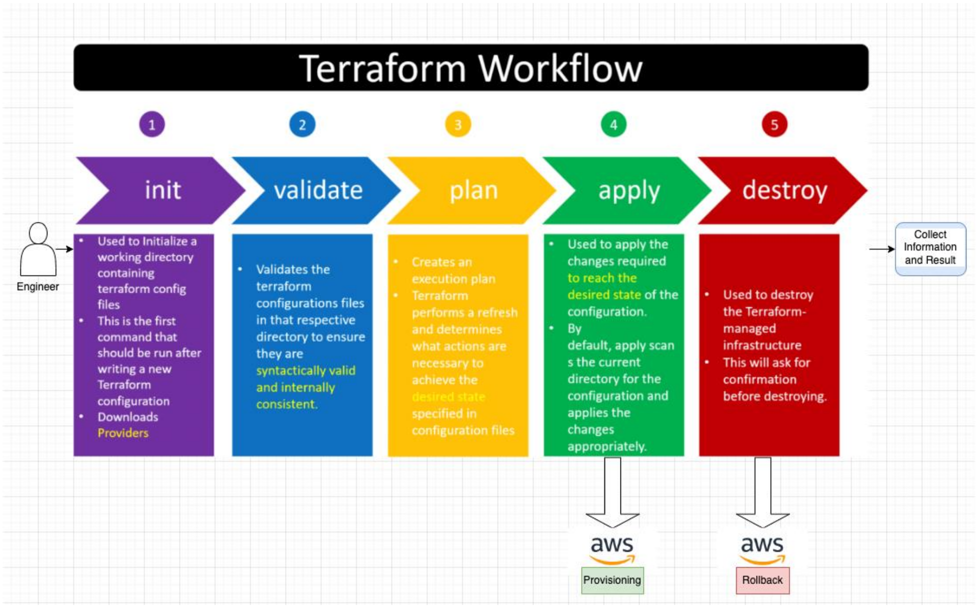
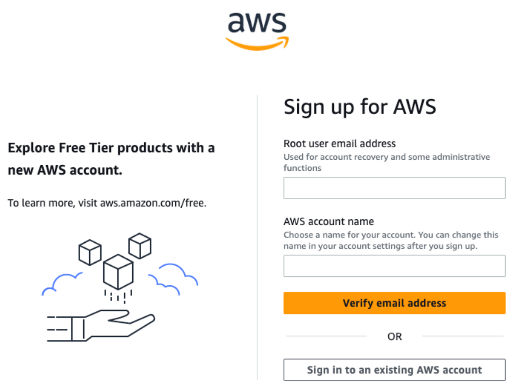

Terraform Workflow
1
2
3
4
5
init
validate
plan
apply
destroy
Engineer
• Used to Initialize a working directory containing terraform config files
• This is the first command that should be run after writing a new Terraform configuration
• Downloads Providers
• Validates the terraform configurations files in that respective directory to ensure they are syntactically valid and internally consistent.
• Creates an execution plan
• Terraform performs a refresh and determines what actions are necessary to achieve the desired state specified in configuration files
• Used to apply the changes required to reach the desired state of the configuration.
• By default, apply scans the current directory for the configuration and applies the changes appropriately.
• Used to destroy the Terraform-managed infrastructure
• This will ask for confirmation before destroying.
Collect Information and Result
aws
Provisioning
aws
Rollback

Figure 3.3 Terraform Workflow

Source: https://terraformguru.com/terraform-real-world-on-azure-cloud/03-Terraform Command-Basics

From Figure 3.2 and 3.3. We will be creating the terraform module and execute the terraform script to provision the resource on AWS.

3.3.1.1 Prerequisites, to have terraform to communicate to AWS. The must do steps are having AWS account, AWS CLI Installed, and AWS credential configured locally.


aws

Explore Free Tier products with a new AWS account.

To learn more, visit aws.amazon.com/free.

Sign up for AWS

Root user email address
Used for account recovery and some administrative functions

AWS account name
Choose a name for your account. You can change this name in your account settings after you sign up.

Verify email address

OR

Sign in to an existing AWS account

Figure 3.4 Create AWS Account on AWS console

From Figure 3.4, go to https://portal.aws.amazon.com/billing/signup#/start/email to create a new AWS account using work or personal email.

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
$ sudo installer -pkg AWSCLIV2.pkg -target /
$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
$ sudo installer -pkg ./AWSCLIV2.pkg -target /
```

Figure 3.5 Install AWS CLI Command

From Figure 3.5, using command line tools to install AWS CLI by following each command.

```
$ aws configure
AWS Access Key ID : XXXXXXXXX
AWS Secret Access Key : XXXXXXXXX
Default region name : ap-southeast-1
```

Figure 3.6 AWS Credential Configure

From Figure 3.6 using command “aws configure” to configure the credentials. The AWS Access Key ID and AWS Secret Access Key were generated when the AWS have been created. Download it and enter the same value. For Default region name, enter the preferred AWS region

3.3.1.2 Create Terraform VPC Module file, which include 2 files.

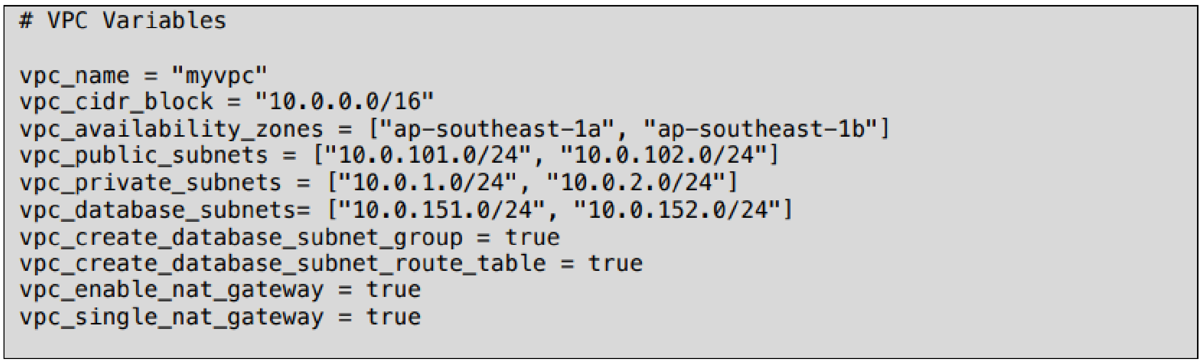
vpc-module.tf, this script will be accommodating the creation of VPC environment on cloud that will include VPC itself, Public and Private Subnet, Nat Gateway, Internet Gateway and routing destination for each subnet.

```
# Create VPC Terraform Module
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "2.78.0"
  #version = "~> 2.78"
  # VPC Basic Details
  name = "${local.name}-${var.vpc_name}"
  cidr = var.vpc_cidr_block
  azs             = var.vpc_availability_zones
  public_subnets  = var.vpc_public_subnets
  private_subnets = var.vpc_private_subnets

  # Database Subnets
  database_subnets = var.vpc_database_subnets
  create_database_subnet_group = var.vpc_create_database_subnet_group
  create_database_subnet_route_table = var.vpc_create_database_subnet_route_table
```

```

  # NAT Gateways - Outbound Communication
  enable_nat_gateway = var.vpc_enable_nat_gateway
  single_nat_gateway = var.vpc_single_nat_gateway

  # VPC DNS Parameters
  enable_dns_hostnames = true
  enable_dns_support   = true
  tags = local.common_tags
  vpc_tags = local.common_tags

  # Additional Tags to Subnets

  public_subnet_tags = {
    Type = "Public Subnets"
  }
  private_subnet_tags = {
    Type = "Private Subnets"
  }
  database_subnet_tags = {
    Type = "Private Database Subnets"
  }
}
```

Figure 3.7 vpc-module.tf file

vpc-variables.tf, this script will collect all the variables that will be using by the module.



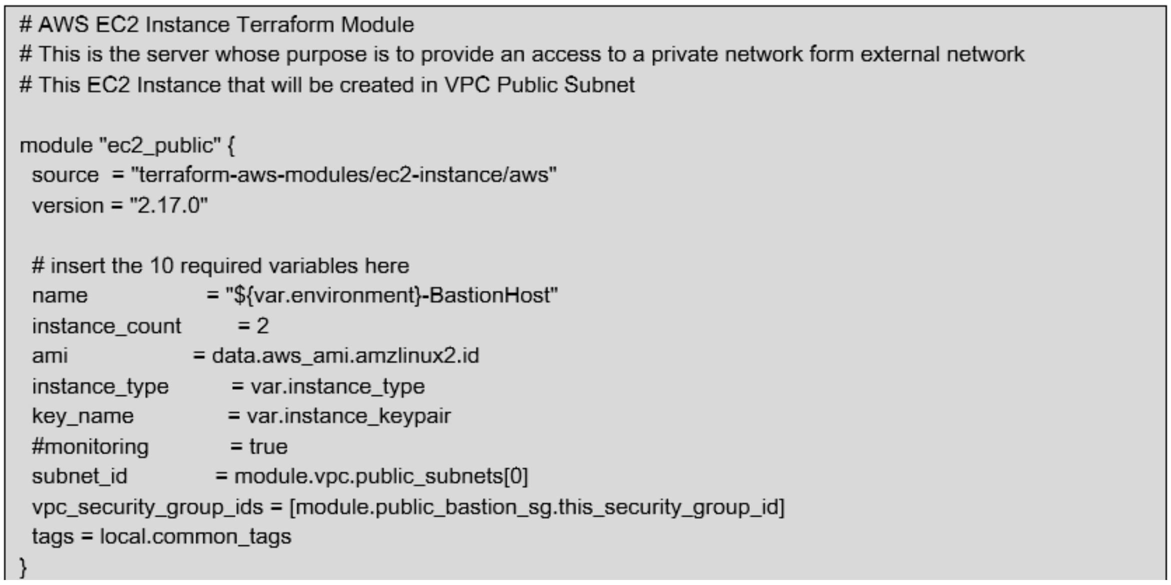
```
# VPC Variables

vpc_name = "myvpc"
vpc_cidr_block = "10.0.0.0/16"
vpc_availability_zones = ["ap-southeast-1a", "ap-southeast-1b"]
vpc_public_subnets = ["10.0.101.0/24", "10.0.102.0/24"]
vpc_private_subnets = ["10.0.1.0/24", "10.0.2.0/24"]
vpc_database_subnets= ["10.0.151.0/24", "10.0.152.0/24"]
vpc_create_database_subnet_group = true
vpc_create_database_subnet_route_table = true
vpc_enable_nat_gateway = true
vpc_single_nat_gateway = true
```

Figure 3.8 vpc-variables.tf file

vpc-output.tf, this script will create a display of an output value after all the creation of the VPC module succeed. This will be used to verify if all the VPC module resource has been created correctly.

3.3.1.3 Create Terraform EC2 Module file, which includes 3 files.

ec2-public.tf, this script will create two EC2 instances, one in each availability zone. They will host the web server and both machines will have an access in and out of the internet through internet gateway.



```
# AWS EC2 Instance Terraform Module
# This is the server whose purpose is to provide an access to a private network form external network
# This EC2 Instance that will be created in VPC Public Subnet

module "ec2_public" {
  source  = "terraform-aws-modules/ec2-instance/aws"
  version = "2.17.0"

  # insert the 10 required variables here
  name                   = "${var.environment}-BastionHost"
  instance_count         = 2
  ami                    = data.aws_ami.amzlinux2.id
  instance_type          = var.instance_type
  key_name               = var.instance_keypair
  #monitoring            = true
  subnet_id              = module.vpc.public_subnets[0]
  vpc_security_group_ids = [module.public_bastion_sg.this_security_group_id]
  tags = local.common_tags
}
```

Figure 3.9 ec2-public.tf file

ec2-private.tf, this script will create two EC2 instances, one in each availability zone. They will host the application server and both machines can communicate with the private subnet. However, they can also go out to the internet by using Nat Gateway service that already create on the VPC model file.

```
# AWS EC2 Instance Terraform Module
# EC2 Instances that will be created in VPC Private Subnets
module "ec2_private" {
  depends_on = [ module.vpc ] # VPC Module need to exist first before this can be create
  source  = "terraform-aws-modules/ec2-instance/aws"
  version = "2.17.0"

  # insert the 10 required variables here
  name                  = "${var.environment}-vm"
  ami                  = data.aws_ami.amzlinux2.id
  instance_type         = var.instance_type
  key_name             = var.instance_keypair
  #monitoring           = true
  vpc_security_group_ids = [module.private_sg.this_security_group_id]
  #subnet_id            = module.vpc.public_subnets[0]
  subnet_ids = [module.vpc.private_subnets[0], module.vpc.private_subnets[1] ]
  instance_count        = var.private_instance_count
  user_data = file("${path.module}/app1-install.sh")
  tags = local.common_tags
}
```

Figure 3.10 ec2-private.tf file

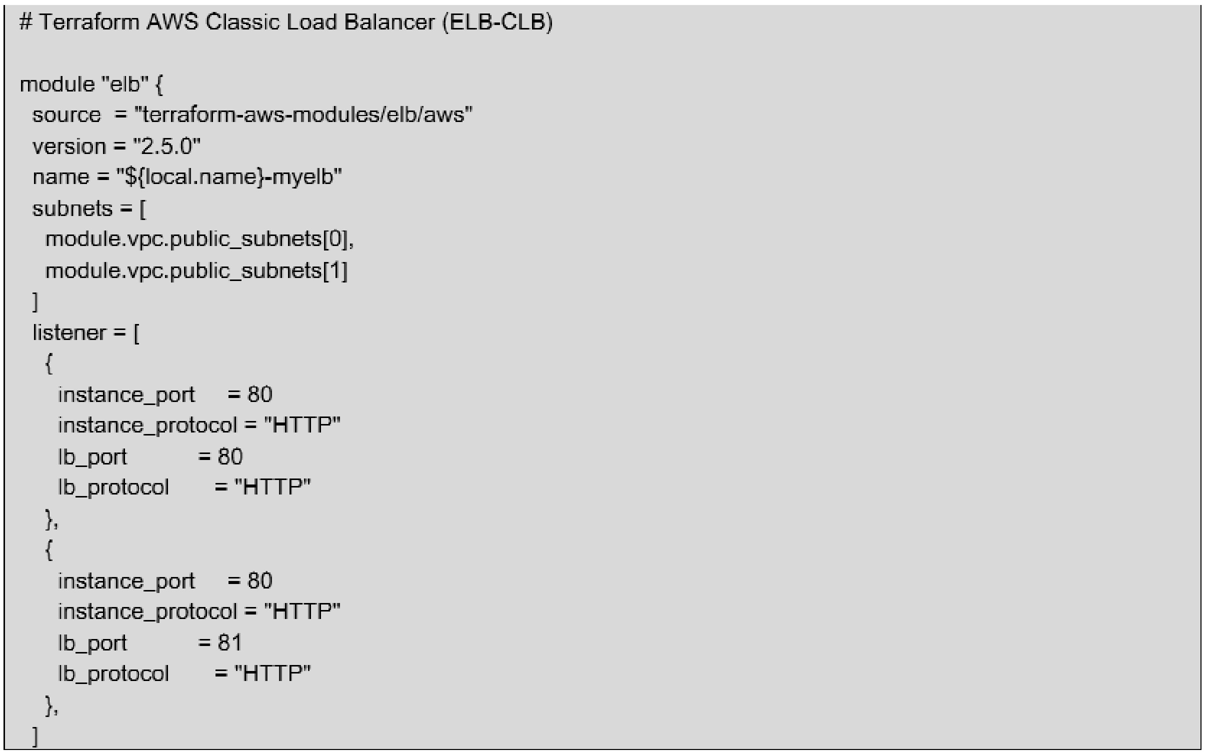
ec2-variables.tf, this script will collect all the variables that will be using by the EC2 module.

```
# EC2 Instance Variables
instance_type = "t2.micro"
instance_keypair = "terraform-key"
private_instance_count = 2
```

Figure 3.11 ec2-variable.tf file

3.3.1.4 Create Terraform Load Balancer Module file

ELB-classic-loadbalancer.tf, this script Load balancer Module will be used for direct the traffic between two EC2 public instance in two availability zones.



```
# Terraform AWS Classic Load Balancer (ELB-CLB)

module "elb" {
  source  = "terraform-aws-modules/elb/aws"
  version = "2.5.0"
  name = "${local.name}-myelb"
  subnets = [
    module.vpc.public_subnets[0],
    module.vpc.public_subnets[1]
  ]
  listener = [
    {
      instance_port     = 80
      instance_protocol = "HTTP"
      lb_port           = 80
      lb_protocol       = "HTTP"
    },
    {
      instance_port     = 80
      instance_protocol = "HTTP"
      lb_port           = 81
      lb_protocol       = "HTTP"
    },
  ]
```

Figure 3.12 ELB-classic-loadbalancer.tf file

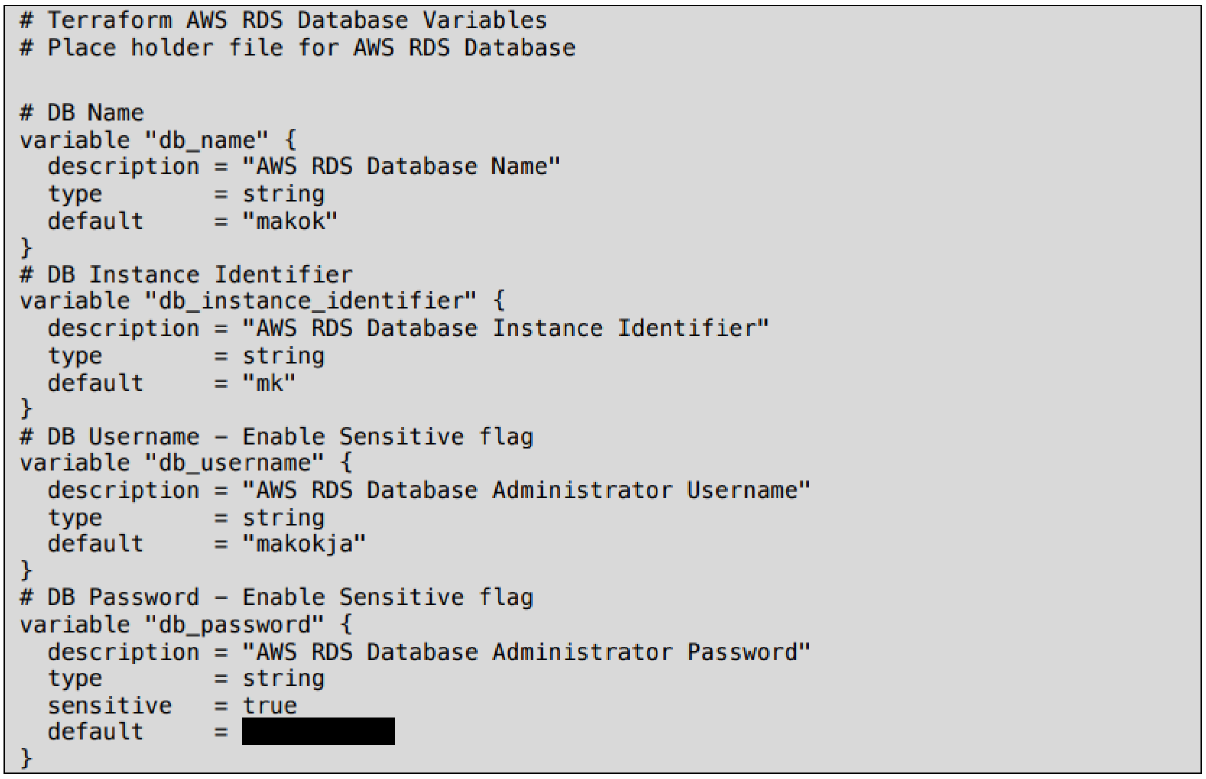
3.3.1.5 Create Terraform RDS Module file, The Amazon RDS id the database service on AWS. In this research we will be using RDS with MySQL database. This will include 2 files for this module.

rdsdb.tf file, this model script will create RDS with MySQL database version 8.0. It will have only one instance and will be configuring with all the database information

```
# Create AWS RDS Database
module "rdsdb" {
  source  = "terraform-aws-modules/rds/aws"
  #version = "2.34.0"
  version = "3.0.0"

  identifier = var.db_instance_identifier
  name     = var.db_name  # Initial Database Name
  username = var.db_username
  password = var.db_password
  port     = 3306
  multi_az               = true
  subnet_ids             = module.vpc.database_subnets
  vpc_security_group_ids = [module.rdsdb_sg.security_group_id]

  # All available versions:
http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_MySQL.html#MySQL.Concept
s.VersionMgmt
  engine               = "mysql"
  engine_version       = "8.0.31"
  family               = "mysql8.0" # DB parameter group
  major_engine_version = "8.0"      # DB option group
  instance_class       = "db.t3.large"

  allocated_storage     = 20
  max_allocated_storage = 100
  storage_encrypted     = false
  maintenance_window              = "Mon:00:00-Mon:03:00"
  backup_window                   = "03:00-06:00"
  enabled_cloudwatch_logs_exports = ["general"]

  backup_retention_period = 0
  skip_final_snapshot     = true
  deletion_protection     = false

  performance_insights_enabled          = true
  performance_insights_retention_period = 7
  create_monitoring_role                = true
  monitoring_interval                   = 60

  parameters = [
    {
      name  = "character_set_client"
      value = "utf8mb4"
    },
    {
      name  = "character_set_server"
      value = "utf8mb4"
    }
  ]

  tags = local.common_tags
  db_instance_tags = {
    "Sensitive" = "high"
  }
  db_option_group_tags = {
    "Sensitive" = "low"
  }
  db_parameter_group_tags = {
    "Sensitive" = "low"
  }
  db_subnet_group_tags = {
    "Sensitive" = "high"
  }
}
```

Figure 3.13 rdsdb.tf file

rds-variables.tf, this script will collect all the variables that will be using by the RDS module.



```
# Terraform AWS RDS Database Variables
# Place holder file for AWS RDS Database

# DB Name
variable "db_name" {
  description = "AWS RDS Database Name"
  type        = string
  default     = "makok"
}
# DB Instance Identifier
variable "db_instance_identifier" {
  description = "AWS RDS Database Instance Identifier"
  type        = string
  default     = "mk"
}
# DB Username - Enable Sensitive flag
variable "db_username" {
  description = "AWS RDS Database Administrator Username"
  type        = string
  default     = "makokja"
}
# DB Password - Enable Sensitive flag
variable "db_password" {
  description = "AWS RDS Database Administrator Password"
  type        = string
  sensitive   = true
  default     = [redacted]
}
```

Figure 3.14 rds-varibles.tf file

3.3.1.6 Create Terraform Security Group Module file, this module will create the security group for EC2, Load Balancer and RDS. The security group are group of the rules that will be determine on who which IP address can be access to each resource on the environment. This module will include 4 files.

securitygroup-loadbalancersg.tf, this script will create the rule to allow who can access the Load Balancer. In this research, the rule will allow any IP addresses to access via port 81.

```
# Security Group for Public Load Balancer
module "loadbalancer_sg" {
  source  = "terraform-aws-modules/security-group/aws"
  version = "3.18.0"

  name        = "loadbalancer-sg"
  description = "Security group with HTTP port open for everybody (IPv4 CIDR),
egress ports are all world open"
  vpc_id      = module.vpc.vpc_id
  # Ingress Rules & CIDR Block
  ingress_rules = ["http-80-tcp"]
  ingress_cidr_blocks = ["0.0.0.0/0"]
  # Egress Rule - all-all open
  egress_rules = ["all-all"]
  tags = local.common_tags
  # Open to CIDRs blocks (rule or from_port+to_port+protocol+description)
  ingress_with_cidr_blocks = [
    {
      from_port   = 81
      to_port     = 81
      protocol    = 6
      description = "Allow Port 81 from internet"
      cidr_blocks = "0.0.0.0/0"
    },
  ]
}
```

Figure 3.15 securitygroup-loadbalancersg.tf file

securitygroup-privatesg.tf, this script will create the rule to allow who can access the EC2 private application instance. In this research, the rule will allow any subnet in the VCP to access via port 22 and 80.

```
# AWS EC2 Security Group Terraform Module
# Security Group for Private EC2 Instances
module "private_sg" {
  source  = "terraform-aws-modules/security-group/aws"
  version = "3.18.0"

  name = "private-sg"
  description = "Security Group with HTTP & SSH port open for entire VPC Block
(IPv4 CIDR), egress ports are all world open"
  vpc_id = module.vpc.vpc_id
  # Ingress Rules & CIDR Blocks
  ingress_rules = ["ssh-tcp", "http-80-tcp"]
  ingress_cidr_blocks = [module.vpc.vpc_cidr_block]
  # Egress Rule - all-all open
  egress_rules = ["all-all"]
  tags = local.common_tags
}
```

Figure 3.16 securitygroup-privatesg.tf file

securitygroup-publicsg.tf, this script will create the rule to allow who can access the EC2 public instance. In this research, the rule will allow any IP Address to access via port 22 and 80.

```
# AWS EC2 Security Group Terraform Module
# Security Group for Public Bastion Host
module "public_bastion_sg" {
  source  = "terraform-aws-modules/security-group/aws"
  version = "3.18.0"
  name = "public-bastion-sg"
  description = "Security Group with SSH port open for everybody (IPv4 CIDR),
egress ports are all world open"
  vpc_id = module.vpc.vpc_id
  # Ingress Rules & CIDR Blocks
  ingress_rules = ["ssh-tcp", "http-80-tcp"]
  ingress_cidr_blocks = ["0.0.0.0/0"]
  # Egress Rule - all-all open
  egress_rules = ["all-all"]
  tags = local.common_tags}
```

Figure 3.17 securitygroup-publicsg.tf file

securitygroup-rds.tf, this script will create the rule to allow who can access the RDS private application instance. In this research, the rule will allow any subnet in the VPC to access via port 3306.

```
# Security Group for AWS RDS DB
module "rdsdb_sg" {
  source  = "terraform-aws-modules/security-group/aws"
  #version = "3.18.0"
  version = "4.0.0"

  name        = "rdsdb-sg"
  description = "Access to MySQL DB for entire VPC CIDR Block"
  vpc_id      = module.vpc.vpc_id

  # ingress
  ingress_with_cidr_blocks = [
    {
      from_port   = 3306
      to_port     = 3306
      protocol    = "tcp"
      description = "MySQL access from within VPC"
      cidr_blocks = module.vpc.vpc_cidr_block
    },
  ]
  # Egress Rule - all-all open
  egress_rules = ["all-all"]
  tags = local.common_tags
}
```

Figure 3.18 securitygroup-rds.tf file

3.3.2 Cloud Formation stack implementation, in this step Cloud Formation stack scripts will be created using yml format file, to build a 3tier web application which consist with all the infrastructure module script in it.

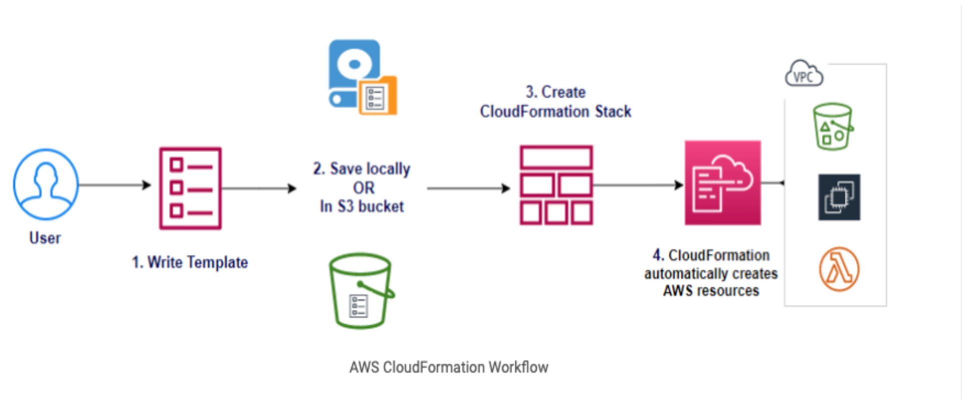
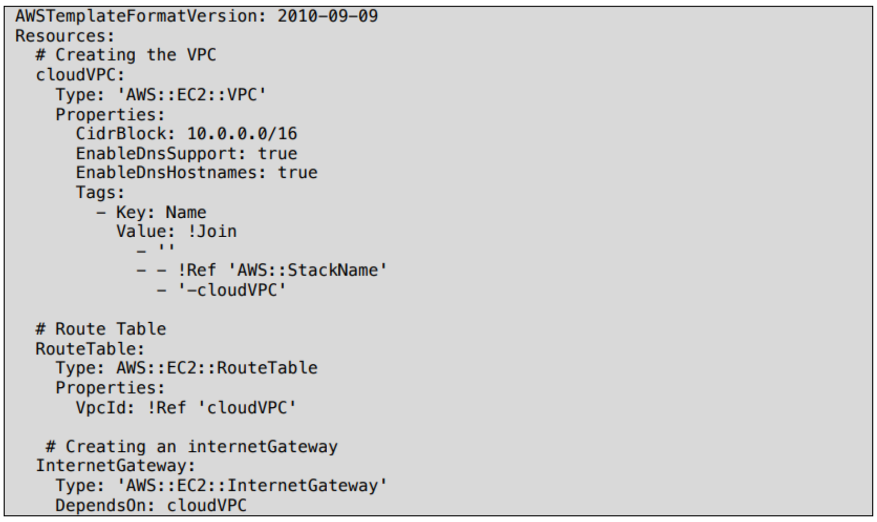


Figure 3.19 Cloud Formation Workflow

3.3.2.1 Create VPC Cloud Formation Template

vpc.yml, this script will be accommodating the creation ofVPC environment on cloud that will include VPC itself, 4 EC2 Instances (2 for public and 2 for private), Public and Private Subnet, Internet Gateway, security groups for the VPC and and routing destination for each subnet. In this file, in order to create EC2, the researcher choose to use the autoscaling group to create EC2 instead of defining the EC2 stack.



```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  # Creating the VPC
  cloudVPC:
    Type: 'AWS::EC2::VPC'
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Join
            - ''
            - - !Ref 'AWS::StackName'
              - '-cloudVPC'

  # Route Table
  RouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref 'cloudVPC'

   # Creating an internetGateway
  InternetGateway:
    Type: 'AWS::EC2::InternetGateway'
    DependsOn: cloudVPC
```

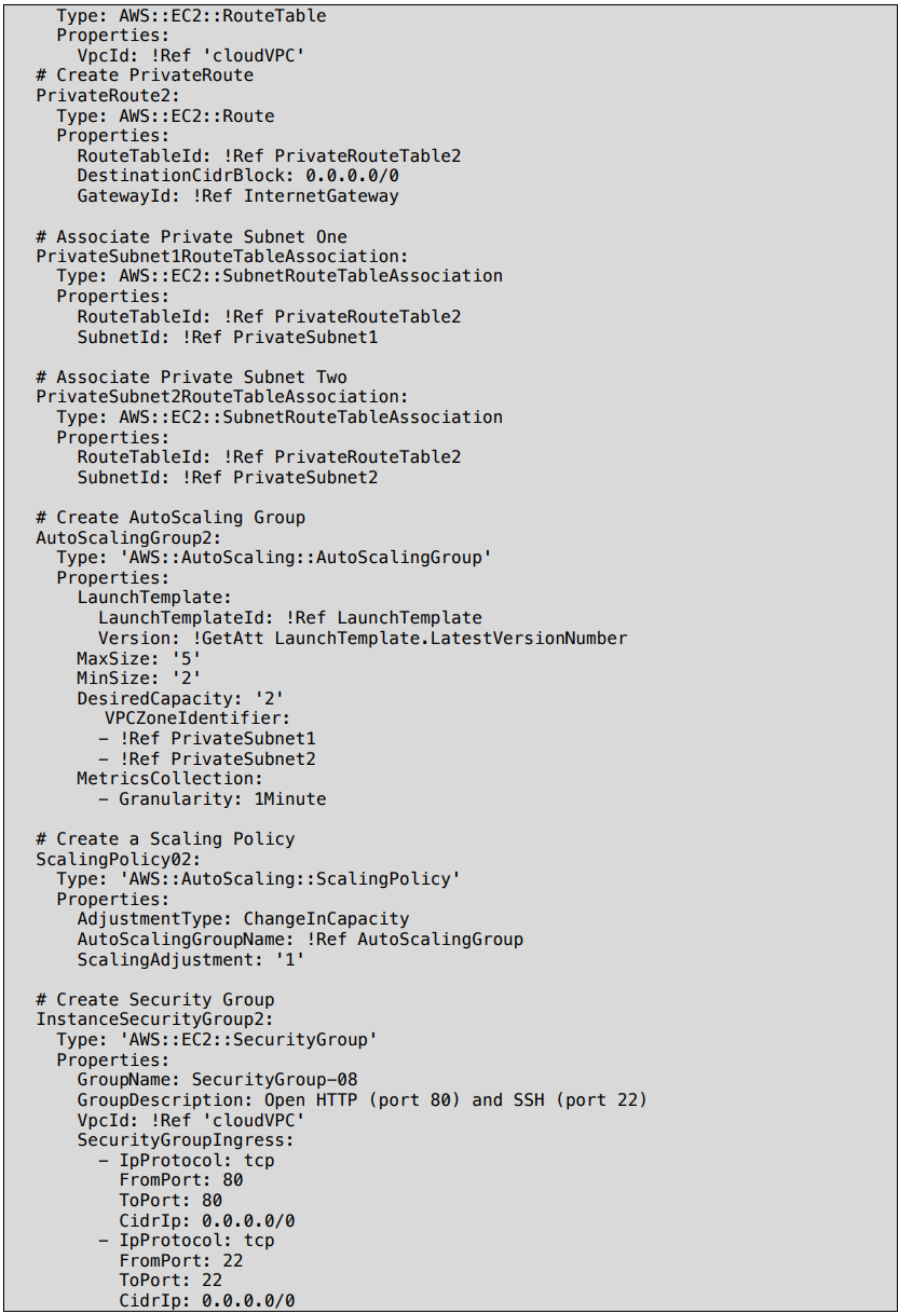
Figure 3.20 vpc,yml file

```
# InternetGatewayAttachment to VPC
  InternetGatewayAttachment:
    Type: 'AWS::EC2::VPCGatewayAttachment'
    Properties:
 # Create Public Subnet One
  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref 'cloudVPC'
      AvailabilityZone: ap-southeast-1a
      CidrBlock: 10.0.101.0/24
      MapPublicIpOnLaunch: true
  # Create Public Subnet Two
  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref 'cloudVPC'
      AvailabilityZone: ap-southeast-1b
      CidrBlock: 10.0.102.0/24
      MapPublicIpOnLaunch: true

  # Create Route Table
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref 'cloudVPC'

  # Create PublicRoute
  PublicRoute:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  # Associate Public Subnet One
  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  # Associate Public Subnet Two
  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2

  # Create Security Group
  InstanceSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupName: SecurityGroup-07
      GroupDescription: Open HTTP (port 80) and SSH (port 22)
      VpcId: !Ref 'cloudVPC'
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
```

Figure 3.20 vpc.yml file **(continue)**

```
# Create Launch Template
  LaunchTemplate:
    Type: 'AWS::EC2::LaunchTemplate'
    Properties:
  # Create AutoScaling Group
  AutoScalingGroup:
    Type: 'AWS::AutoScaling::AutoScalingGroup'
    Properties:
      LaunchTemplate:
        LaunchTemplateId: !Ref LaunchTemplate
        Version: !GetAtt LaunchTemplate.LatestVersionNumber
      MaxSize: '5'
      MinSize: '2'
      DesiredCapacity: '2'
      VPCZoneIdentifier:
        - !Ref PublicSubnet1
        - !Ref PublicSubnet2
      MetricsCollection:
        - Granularity: 1Minute

  # Create a Scaling Policy
  ScalingPolicy07:
    Type: 'AWS::AutoScaling::ScalingPolicy'
    Properties:
      AdjustmentType: ChangeInCapacity
      AutoScalingGroupName: !Ref AutoScalingGroup
      ScalingAdjustment: '1'

  # Create Private Subnet One
  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref 'cloudVPC'
      AvailabilityZone: ap-southeast-1a
      CidrBlock: 10.0.1.0/24
      MapPublicIpOnLaunch: false

  # Create Private Subnet Two
  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref 'cloudVPC'
      AvailabilityZone: ap-southeast-1b
      CidrBlock: 10.0.2.0/24
      MapPublicIpOnLaunch: false

  # Create Private Subnet Three
  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref 'cloudVPC'
      AvailabilityZone: ap-southeast-1a
      CidrBlock: 10.0.151.0/24
      MapPublicIpOnLaunch: false

  # Create Private Subnet Four
  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref 'cloudVPC'
      AvailabilityZone: ap-southeast-1b
      CidrBlock: 10.0.152.0/24
      MapPublicIpOnLaunch: false
   # Create Private Route Table
    PrivateRouteTable2
```

Figure 3.20 vpc.yml file **(continue)**



```
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref 'cloudVPC'
  # Create PrivateRoute
  PrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  # Associate Private Subnet One
  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet1

  # Associate Private Subnet Two
  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  # Create AutoScaling Group
  AutoScalingGroup2:
    Type: 'AWS::AutoScaling::AutoScalingGroup'
    Properties:
      LaunchTemplate:
        LaunchTemplateId: !Ref LaunchTemplate
        Version: !GetAtt LaunchTemplate.LatestVersionNumber
      MaxSize: '5'
      MinSize: '2'
      DesiredCapacity: '2'
         VPCZoneIdentifier:
        - !Ref PrivateSubnet1
        - !Ref PrivateSubnet2
      MetricsCollection:
        - Granularity: 1Minute

  # Create a Scaling Policy
  ScalingPolicy02:
    Type: 'AWS::AutoScaling::ScalingPolicy'
    Properties:
      AdjustmentType: ChangeInCapacity
      AutoScalingGroupName: !Ref AutoScalingGroup
      ScalingAdjustment: '1'

  # Create Security Group
  InstanceSecurityGroup2:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupName: SecurityGroup-08
      GroupDescription: Open HTTP (port 80) and SSH (port 22)
      VpcId: !Ref 'cloudVPC'
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
```

Figure 3.20 vpc,yml file **(continue)**

3.3.2.1 Create Amazon RDS CloudFormation Template

database.yml, this template will create RDS with MySQL database version 8.0. It will have only 1 instance and will be configuring with all the database information. In this template there will be some parameter that does not define yet, as the limitation of the CloudFormation is that the resource ID will never be known. If the VPC itself has not been cerate yet. So, the researcher decides to use parameter with drop-down list and then it can be chosen when creating this stack after the VPC Stack has been create

```
#Create an RDS MySql Instance
Parameters:
  Owner:
    Description: Enter the Name of the owner for this Stack.
    Type: String
    Default: Name
  VPC:
    Description: Select VPC form the available VPCs in your account.
    Type: AWS::EC2::VPC::Id
  PrivateSubnet1:
    Description: Select Private Subnet 1.
    Type: AWS::EC2::Subnet::Id
  PrivateSubnet2:
    Description: Select Private Subnet 2.
    Type: AWS::EC2::Subnet::Id
  MasterUsername:
    Description: Database administration name.
    Type: String
    Default: rdsroot
  MasterUserPassword:
    NoEcho: 'true'
    Description: Database administration password.
    Type: String
    MinLength: '8'
    AllowedPattern: "[a-zA-Z0-9!?]*"
    ConstraintDescription: Must only contain upper and lowercase and numbers
  MultiAvailabilityZone:
    Description: Do you want to Enable Multi Availability Zones?
    Type: String
    Default: 'true'
    AllowedValues:
    - 'true'
    - 'false'
  TcpPort:
    Description: Enter RDS Listening TCP Port number.
    Type: Number
    Default: '3306'
  AutoMinorVersionUpgrade:
    Description: Do you want to allow automatic minor version upgrade?
    Type: String
    Default: 'true'
    AllowedValues:
    - 'true'
    - 'false'
  InstanceType:
    Description: Select Instance Type.
    Type: String
```

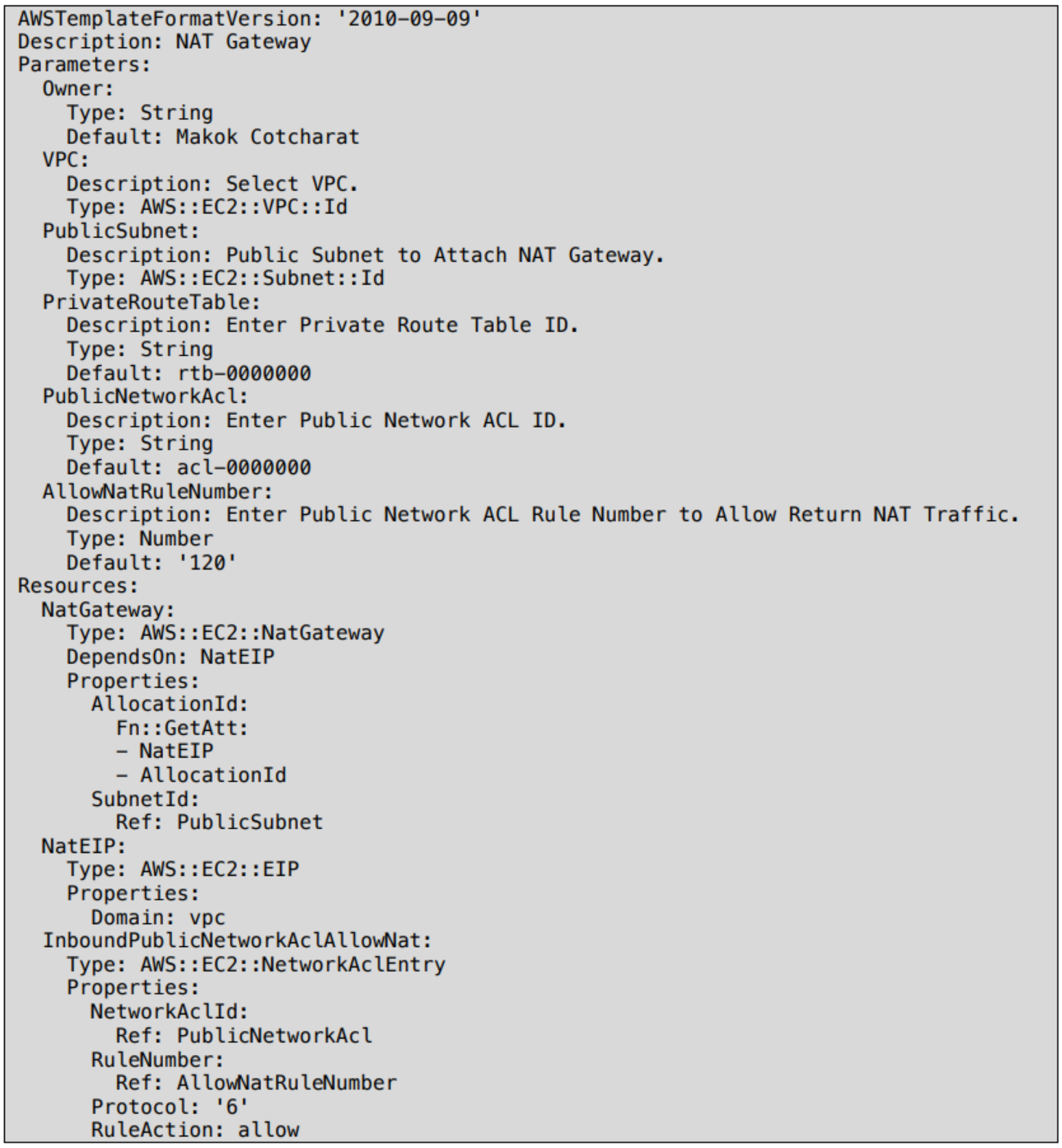
Figure 3.21 Database.yml file

```
    Default: db.t2.micro
    ConstraintDescription: Must be a valid EC2 instance type.

Mappings:
  Settings:
    MySQL:
      Engine: MySQL
      Version: '8.0'

Conditions:
  ConfigureSnapshotOnDelete:
    Fn::Equals:
    - Ref: SnapshotOnDelete
    - 'true'

Resources:
  RDSAccessSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Instance to RDS Access
      VpcId:
        Ref: VPC
      Tags:
      - Key: Name
        Value:
          Fn::Join:
          - ''
          - - Ref: AWS::StackName
            - "-rds"
      - Key: Owner
        Value:
          Ref: Owner

  AccessSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    DependsOn: RDSAccessSecurityGroup
    Properties:
      GroupId:
        Ref: RDSAccessSecurityGroup
      IpProtocol: tcp
      FromPort:
        Ref: TcpPort
      ToPort:
        Ref: TcpPort
      SourceSecurityGroupId:
        Ref: RDSAccessSecurityGroup

  DbSubnetGroup:
    Type: AWS::RDS::DBSubnetGroup
    Properties:
      DBSubnetGroupDescription:
        Fn::Join:
        - ''
        - - 'RDS Subnet Group for '
          - Ref: AWS::StackName
      SubnetIds:
      - Ref: PrivateSubnet1
      - Ref: PrivateSubnet2
      Tags:
      - Key: Name
        Value:
          Ref: AWS::StackName
      - Key: Owner
        Value:
          Ref: Owner
```

Figure 3.21 Database.yml file **(Continue)**

3.3.2.1 Create Nat Gate CloudFormation Template

natgateway.yml, this template will build the Nat Gateway that will allow the 2 private EC2 instances that host the application to access the internet as they might need to connect with APIs, but it will not allow any one from outside access these instances directly. The reason we need to separate the template of the Nat Gateway because when we build the VPC and all the subnets. We still do not have the subnet ID that need to be accommodate with Nat Gateway. That is why we need to build the VPC first then build the Nat Gateway after.



```
AWSTemplateFormatVersion: '2010-09-09'
Description: NAT Gateway
Parameters:
  Owner:
    Type: String
    Default: Makok Cotcharat
  VPC:
    Description: Select VPC.
    Type: AWS::EC2::VPC::Id
  PublicSubnet:
    Description: Public Subnet to Attach NAT Gateway.
    Type: AWS::EC2::Subnet::Id
  PrivateRouteTable:
    Description: Enter Private Route Table ID.
    Type: String
    Default: rtb-0000000
  PublicNetworkAcl:
    Description: Enter Public Network ACL ID.
    Type: String
    Default: acl-0000000
  AllowNatRuleNumber:
    Description: Enter Public Network ACL Rule Number to Allow Return NAT Traffic.
    Type: Number
    Default: '120'
Resources:
  NatGateway:
    Type: AWS::EC2::NatGateway
    DependsOn: NatEIP
    Properties:
      AllocationId:
        Fn::GetAtt:
        - NatEIP
        - AllocationId
      SubnetId:
        Ref: PublicSubnet
  NatEIP:
    Type: AWS::EC2::EIP
    Properties:
      Domain: vpc
  InboundPublicNetworkAclAllowNat:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
      NetworkAclId:
        Ref: PublicNetworkAcl
      RuleNumber:
        Ref: AllowNatRuleNumber
      Protocol: '6'
      RuleAction: allow
```

Figure 3.22 natgateway.yml file

```
        CidrBlock: 0.0.0.0/0
      PortRange:
        From: '1024'
        To: '65535'
Outputs:
  Owner:
    Description: Team or Individual that Owns this Formation.
    Value:
      Ref: Owner
  VPC:
    Description: VPC Used
    Value:
      Ref: VPC
  NatEIP:
    Description: NAT Elastic IP Address
    Value:
      Ref: NatEIP
  PublicNetworkACLRuleNumbers:
    Description: Public Network ACL Rules Number Created.
    Value:
      Fn::Join:
      - ''
      - - Inbound (
        - Ref: AllowNatRuleNumber
        - ")"
Metadata:
  AWS::CloudFormation::Interface:
    ParameterGroups:
    - Label:
        default: Ownership
      Parameters:
      - Owner
    - Label:
        default: Network Configuration
      Parameters:
      - VPC
      - PublicSubnet
      - PrivateRouteTable
      - PublicNetworkAcl
      - AllowNatRuleNumber
    ParameterLabels:
      Owner:
        default: Team or Individual Owner
      PublicSubnet:
        default: Public Subnet
      PrivateRouteTable:
        default: Private Route Table
      PublicNetworkAcl:
        default: Public Network ACL
      AllowNatRuleNumber:
        default: Public Network ACL Rule Number
```

Figure 3.22 natgateway.yml file **(Continue)**

CHAPTER 4

SYSTEM TESTING AND RESULT

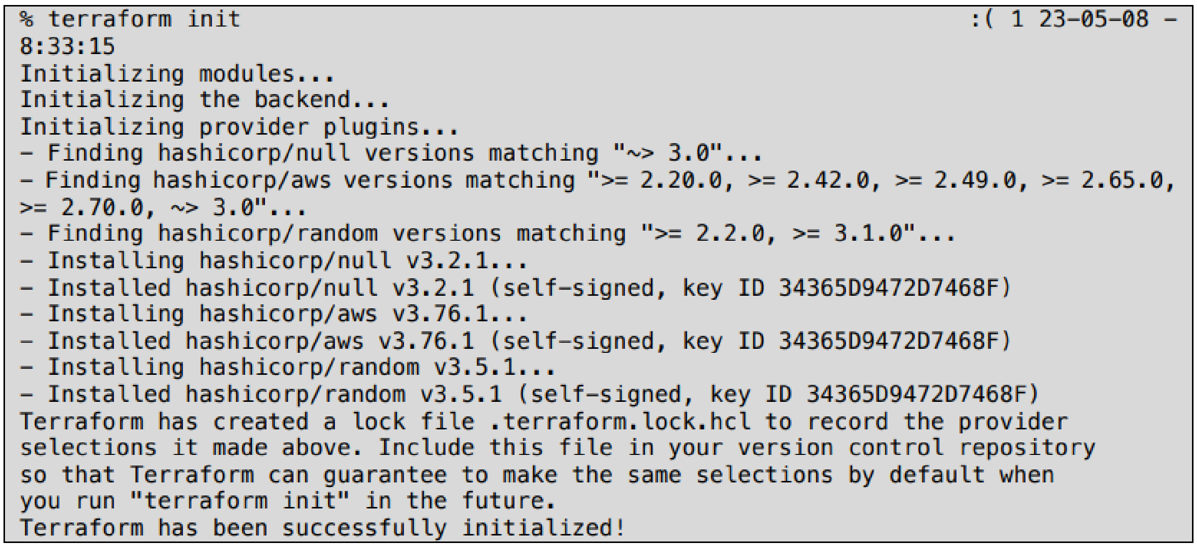
In this chapter, we will discuss about the system testing and result on the comparison between Terraform and CloudFormation stack. The comparation criteria will be as follows.

- Capability
- Efficiency
- Error Handling
- Rollback Ability

4.1 Capability Test

4.1.1 Terraform Capability Test, the capabilities of the terraform will be test by execute the command on the CLI by following the below steps.

4.1.1.1 Run “terraform init” command, this will initialize terraform directory and terraform lock file. Including install provider dependencies plugin.

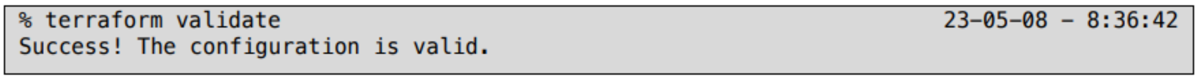


```
% terraform init                                                    :( 1 23-05-08 -
8:33:15
Initializing modules...
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/null versions matching "~> 3.0"...
- Finding hashicorp/aws versions matching ">= 2.20.0, >= 2.42.0, >= 2.49.0, >= 2.65.0,
>= 2.70.0, ~> 3.0"...
- Finding hashicorp/random versions matching ">= 2.2.0, >= 3.1.0"...
- Installing hashicorp/null v3.2.1...
- Installed hashicorp/null v3.2.1 (self-signed, key ID 34365D9472D7468F)
- Installing hashicorp/aws v3.76.1...
- Installed hashicorp/aws v3.76.1 (self-signed, key ID 34365D9472D7468F)
- Installing hashicorp/random v3.5.1...
- Installed hashicorp/random v3.5.1 (self-signed, key ID 34365D9472D7468F)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.
Terraform has been successfully initialized!
```

Figure 4.1 Terraform init

From Figure 4.1 It showed that terraform init command is successful as all the plugin and dependencies file has been created.

4.1.1.2 Run "terraform validate" command, this will validate the terraform script to make sure there are no error for the code.

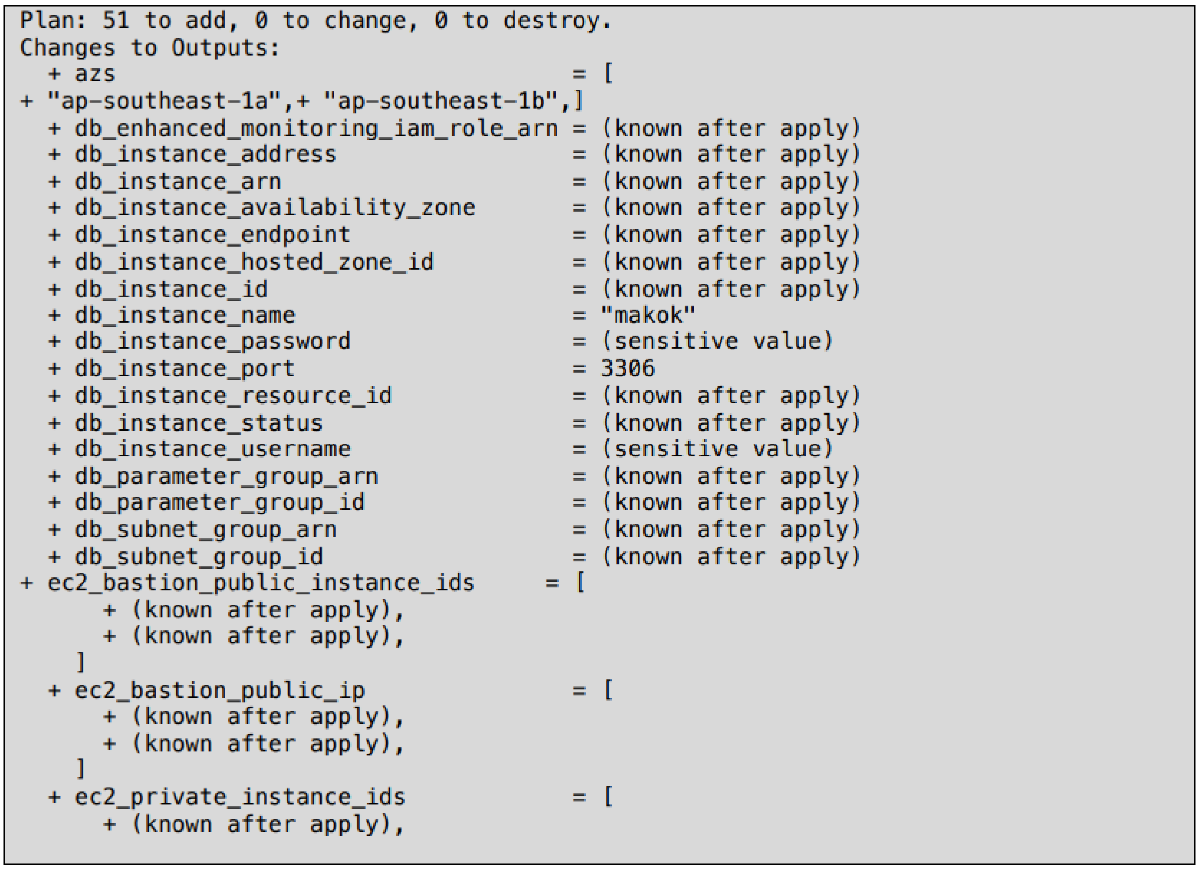


```
% terraform validate                                                    23-05-08 - 8:36:42
Success! The configuration is valid.
```

Figure 4.2 Terraform validate

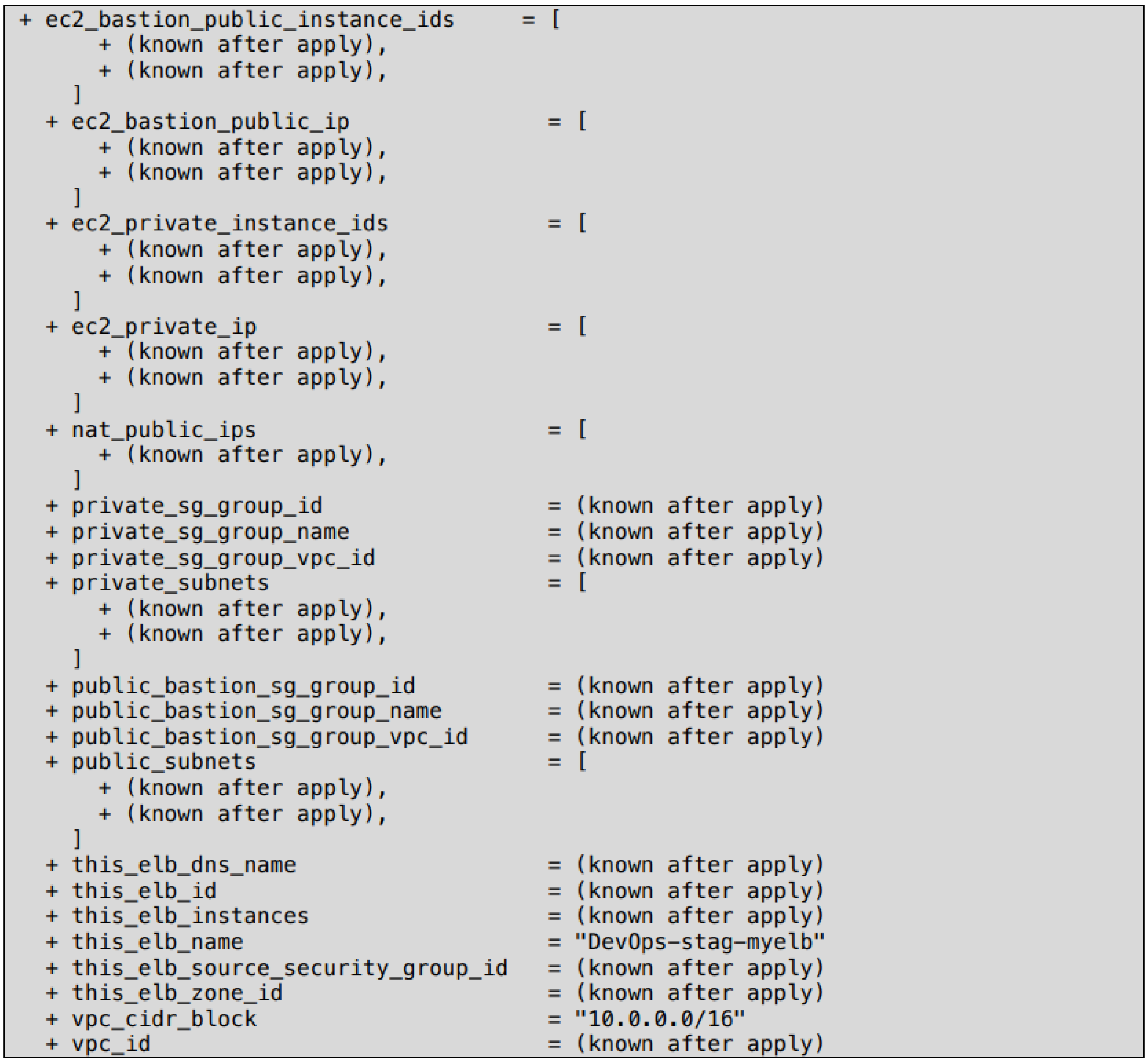
From Figure 4.2 It showed that terraform validate command is successful. Meaning that there is no error on the terraform script and configurations the were created.

4.1.1.3 Run "terraform plan" command, this will give you an execution plan and gives you a chance to review what would change on your infrastructure. It also collects the new desired lock state of the terraform., in case it needs to be rollback.



```
Plan: 51 to add, 0 to change, 0 to destroy.
Changes to Outputs:
  + azs                                  = [
+ "ap-southeast-1a",+ "ap-southeast-1b",]
  + db_enhanced_monitoring_iam_role_arn = (known after apply)
  + db_instance_address                  = (known after apply)
  + db_instance_arn                      = (known after apply)
  + db_instance_availability_zone        = (known after apply)
  + db_instance_endpoint                 = (known after apply)
  + db_instance_hosted_zone_id           = (known after apply)
  + db_instance_id                       = (known after apply)
  + db_instance_name                     = "makok"
  + db_instance_password                 = (sensitive value)
  + db_instance_port                     = 3306
  + db_instance_resource_id              = (known after apply)
  + db_instance_status                   = (known after apply)
  + db_instance_username                 = (sensitive value)
  + db_parameter_group_arn               = (known after apply)
  + db_parameter_group_id                = (known after apply)
  + db_subnet_group_arn                  = (known after apply)
  + db_subnet_group_id                   = (known after apply)
+ ec2_bastion_public_instance_ids      = [
      + (known after apply),
      + (known after apply),
    ]
  + ec2_bastion_public_ip                = [
      + (known after apply),
      + (known after apply),
    ]
  + ec2_private_instance_ids             = [
      + (known after apply),
```

Figure 4.3 terraform plan

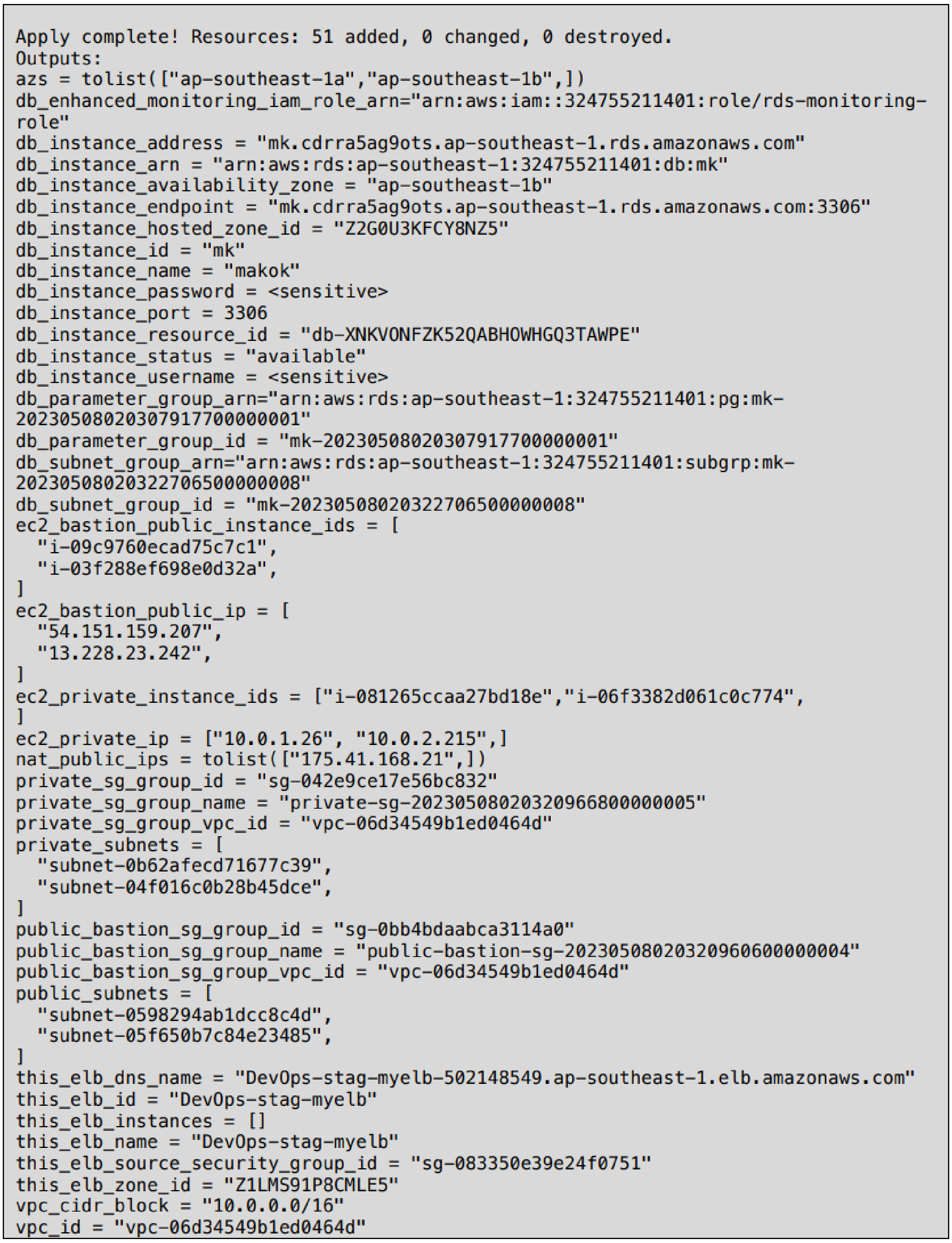


```
+ ec2_bastion_public_instance_ids     = [
      + (known after apply),
      + (known after apply),
    ]
  + ec2_bastion_public_ip               = [
      + (known after apply),
      + (known after apply),
    ]
  + ec2_private_instance_ids            = [
      + (known after apply),
      + (known after apply),
    ]
  + ec2_private_ip                      = [
      + (known after apply),
      + (known after apply),
    ]
  + nat_public_ips                      = [
      + (known after apply),
    ]
  + private_sg_group_id                 = (known after apply)
  + private_sg_group_name               = (known after apply)
  + private_sg_group_vpc_id             = (known after apply)
  + private_subnets                     = [
      + (known after apply),
      + (known after apply),
    ]
  + public_bastion_sg_group_id          = (known after apply)
  + public_bastion_sg_group_name        = (known after apply)
  + public_bastion_sg_group_vpc_id      = (known after apply)
  + public_subnets                      = [
      + (known after apply),
      + (known after apply),
    ]
  + this_elb_dns_name                   = (known after apply)
  + this_elb_id                         = (known after apply)
  + this_elb_instances                  = (known after apply)
  + this_elb_name                       = "DevOps-stag-myelb"
  + this_elb_source_security_group_id   = (known after apply)
  + this_elb_zone_id                    = (known after apply)
  + vpc_cidr_block                      = "10.0.0.0/16"
  + vpc_id                              = (known after apply)
```

Figure 4.3 terraform plan **(Continue)**

From Figure 4.3, It showed that how many resource will be added on to the AWS infrastructure, we can review on each resource that it need to be add.

4.1.1.4 Run “terraform apply” command, this will apply all the change on to AWS. Your AWS resources will get provisioned according to the script.



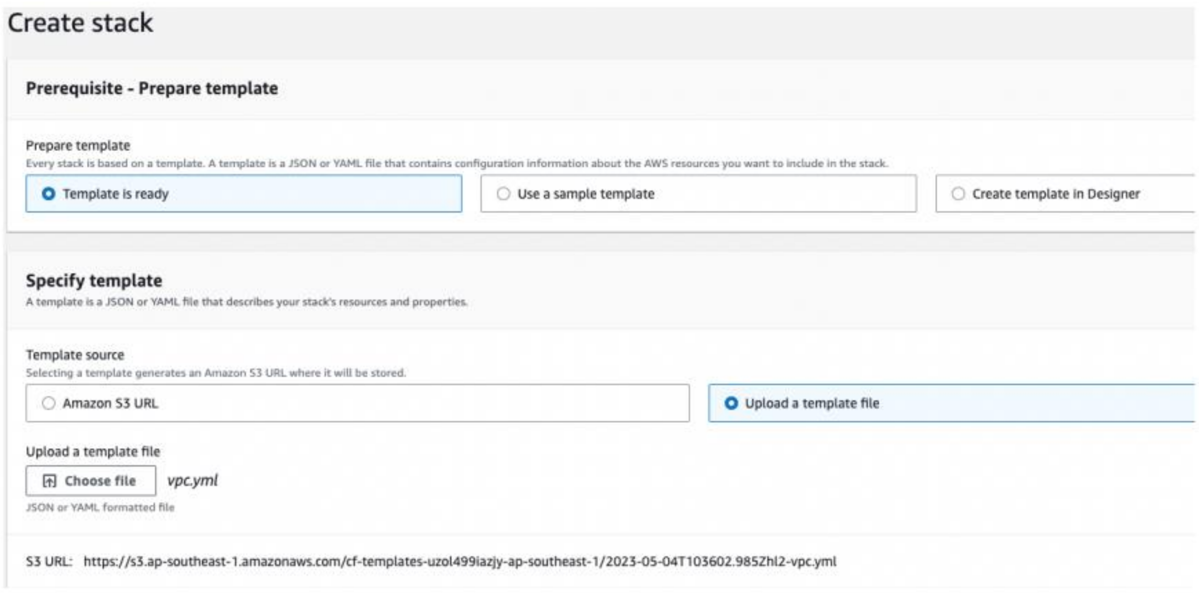
```
Apply complete! Resources: 51 added, 0 changed, 0 destroyed.
Outputs:
azs = tolist(["ap-southeast-1a","ap-southeast-1b",])
db_enhanced_monitoring_iam_role_arn="arn:aws:iam::324755211401:role/rds-monitoring-
role"
db_instance_address = "mk.cdrra5ag9ots.ap-southeast-1.rds.amazonaws.com"
db_instance_arn = "arn:aws:rds:ap-southeast-1:324755211401:db:mk"
db_instance_availability_zone = "ap-southeast-1b"
db_instance_endpoint = "mk.cdrra5ag9ots.ap-southeast-1.rds.amazonaws.com:3306"
db_instance_hosted_zone_id = "Z2G0U3KFCY8NZ5"
db_instance_id = "mk"
db_instance_name = "makok"
db_instance_password = <sensitive>
db_instance_port = 3306
db_instance_resource_id = "db-XNKVONFZK52QABHOWHGQ3TAWPE"
db_instance_status = "available"
db_instance_username = <sensitive>
db_parameter_group_arn="arn:aws:rds:ap-southeast-1:324755211401:pg:mk-
20230508020307917700000001"
db_parameter_group_id = "mk-20230508020307917700000001"
db_subnet_group_arn="arn:aws:rds:ap-southeast-1:324755211401:subgrp:mk-
20230508020322706500000008"
db_subnet_group_id = "mk-20230508020322706500000008"
ec2_bastion_public_instance_ids = [
  "i-09c9760ecad75c7c1",
  "i-03f288ef698e0d32a",
]
ec2_bastion_public_ip = [
  "54.151.159.207",
  "13.228.23.242",
]
ec2_private_instance_ids = ["i-081265ccaa27bd18e","i-06f3382d061c0c774",
]
ec2_private_ip = ["10.0.1.26", "10.0.2.215",]
nat_public_ips = tolist(["175.41.168.21",])
private_sg_group_id = "sg-042e9ce17e56bc832"
private_sg_group_name = "private-sg-20230508020320966800000005"
private_sg_group_vpc_id = "vpc-06d34549b1ed0464d"
private_subnets = [
  "subnet-0b62afecd71677c39",
  "subnet-04f016c0b28b45dce",
]
public_bastion_sg_group_id = "sg-0bb4bdaabca3114a0"
public_bastion_sg_group_name = "public-bastion-sg-20230508020320960600000004"
public_bastion_sg_group_vpc_id = "vpc-06d34549b1ed0464d"
public_subnets = [
  "subnet-0598294ab1dcc8c4d",
  "subnet-05f650b7c84e23485",
]
this_elb_dns_name = "DevOps-stag-myelb-502148549.ap-southeast-1.elb.amazonaws.com"
this_elb_id = "DevOps-stag-myelb"
this_elb_instances = []
this_elb_name = "DevOps-stag-myelb"
this_elb_source_security_group_id = "sg-083350e39e24f0751"
this_elb_zone_id = "Z1LMS91P8CMLE5"
vpc_cidr_block = "10.0.0.0/16"
vpc_id = "vpc-06d34549b1ed0464d"
```

Figure 4.4 terraform apply

From Figure 4.4 It showed the output that all the resources configurations have been provisioning on to AWS.

4.1.1.5 Check on AWS console to see if the infrastructure resources that we are provisioning using terraform are really exists on AWS

4.1.2 Cloud Formation Stack Capabilities Test, create a stack on AWS CloudFormation page, create a stack by upload the template each template file start from "vpc.yml", "database.yml", and "natgateway.yml" in order. The provisioning of the resource will be start after stack has been create. Wait for one to finished before continuing the next one.



Create stack

Prerequisite - Prepare template

Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

[✓ Template is ready] Use a sample template Create template in Designer

Specify template
A template is a JSON or YAML file that describes your stack's resources and properties.

Template source
Selecting a template generates an Amazon S3 URL where it will be stored.

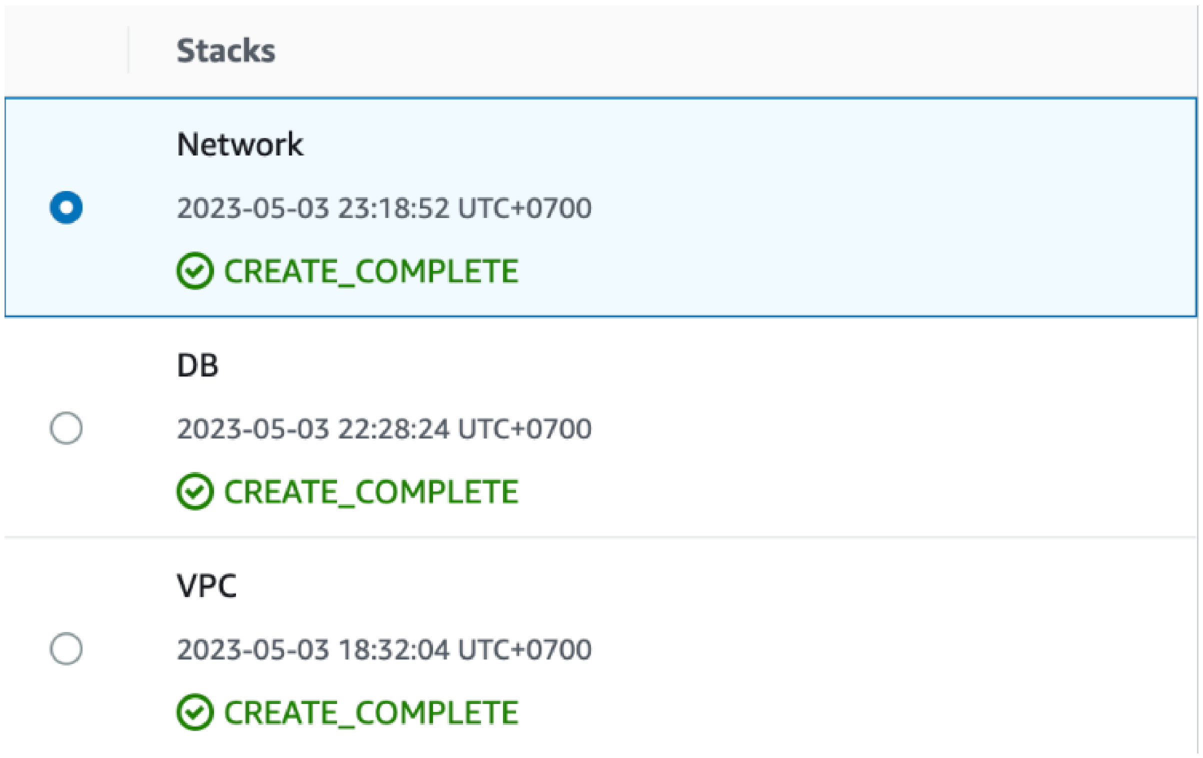
Amazon S3 URL [✓ Upload a template file]

Upload a template file
Choose file *vpc.yml*
JSON or YAML formatted file

S3 URL: https://s3.ap-southeast-1.amazonaws.com/cf-templates-uzol499iazjy-ap-southeast-1/2023-05-04T103602.985Zhl2-vpc.yml

Figure 4.5 Create Cloud Formation Stack

From Figure 4.5, the template needs to be uploaded to create the stack and the CloudFormation stack will run immediately after the stack has been created. To create a stack the template file can only be upload one file at a time. We can run the multiple stacks in parallel hence, each stack is not depending on each other's. If the stack has the dependencies, we must create the stack that other depends on first before creating another stack. For example, in this case the VPC stack need to create first before we can create the database stack because database must have the VPC before it can exist.



Stacks

Network
2023-05-03 23:18:52 UTC+0700
CREATE_COMPLETE

DB
2023-05-03 22:28:24 UTC+0700
CREATE_COMPLETE

VPC
2023-05-03 18:32:04 UTC+0700
CREATE_COMPLETE

Figure 4.6 Cloud Formation Stack Execution Completed

From Figure 4.6, after creating a stack and wait for everything to run. It should show that that all the stacks are complete.

4.1.3 Capability Test Result

After completed the creation of the infrastructure on AWS using Terraform and Cloud Formation stack. The result will be gathered to see if all necessary components that needed for 3 tier web application were built successfully using both tools.

Table 4.1 Result of the capabilities test. Classify by the resource type that needed to be built

Resource Types	Terraform	CloudFormation
VPC and Its Components	✓	✓
Security Group	✓	✓
EC2 (Private and Public)	✓	✓
Load Balancer	✓	✓
Amazon RDS (Database)	✓	✓

Remark: ✓ = Successful ✗ = Unsuccessful

From Table 4.1, All the resources type that need to be provisioned on AWS can be provisioned using both tools, so we cannot conclude on which tools is better in terms of the capabilities test.

4.2 Efficiency Test

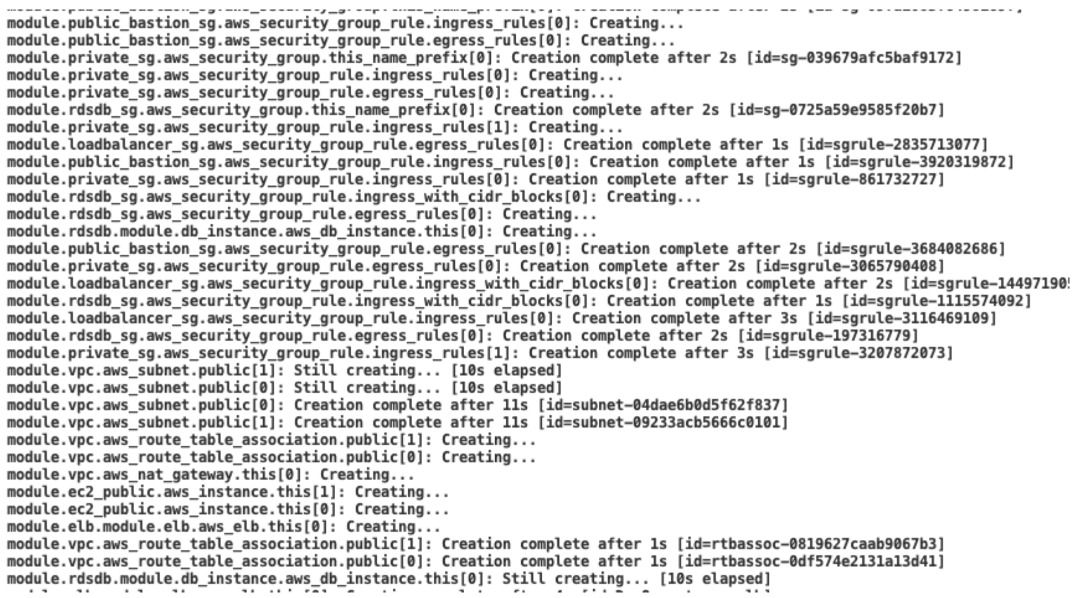
On the efficiency test, we will measure the time for start to finish when we execute all the provisioning tasks and measure the CPU and memory resources that terraform needed during the execution.

- Terraform and CloudFormation Stack need to be run on the same computer.
- The computer spec that uses to run this have 10 Cores CPU and 16 GB of Memory.
- All the programs and application on the computer need to be closed except for the terraform and the CloudFormation AWS web browser.
- The time measurement will be done on each resource type that has been created.

- The computer resources maturement during the executions will be measured during execution time from start to finish.
- CloudFormation Stack cannot be measure for the resources use, because it is AWS manage service. When execute the stack, it does not use any use devices resource. Except for the browser that resource that need to because to open the AWS CloudFormation Stack's URL. The conditions for the measurement here are as follows.

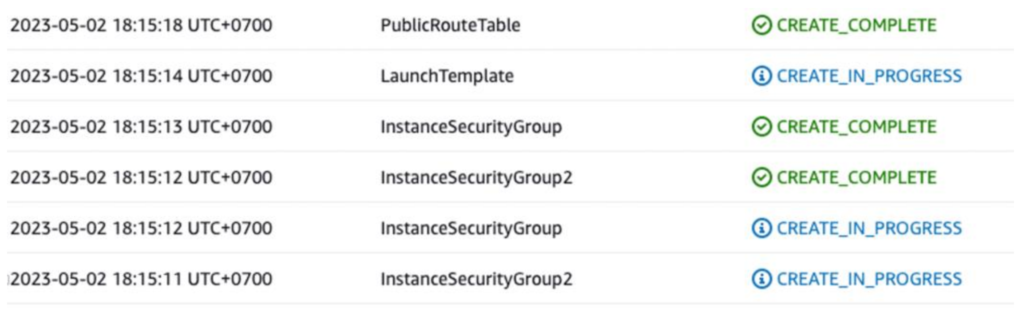
4.2.1 Time Measurement Test Result

The time measurement for both tools can be done easily as both tools has a time stamp for each resource tasks that created as in Figure 4.7 and 4.8



```
module.public_bastion_sg.aws_security_group_rule.ingress_rules[0]: Creating...
module.public_bastion_sg.aws_security_group_rule.egress_rules[0]: Creating...
module.private_sg.aws_security_group.this_name_prefix[0]: Creation complete after 2s [id=sg-039679afc5baf9172]
module.private_sg.aws_security_group_rule.ingress_rules[0]: Creating...
module.private_sg.aws_security_group_rule.egress_rules[0]: Creating...
module.rdsdb_sg.aws_security_group.this_name_prefix[0]: Creation complete after 2s [id=sg-0725a59e9585f20b7]
module.private_sg.aws_security_group_rule.ingress_rules[1]: Creating...
module.loadbalancer_sg.aws_security_group_rule.egress_rules[0]: Creation complete after 1s [id=sgrule-2835713077]
module.public_bastion_sg.aws_security_group_rule.ingress_rules[0]: Creation complete after 1s [id=sgrule-3920319872]
module.private_sg.aws_security_group_rule.ingress_rules[0]: Creation complete after 1s [id=sgrule-861732727]
module.rdsdb_sg.aws_security_group_rule.ingress_with_cidr_blocks[0]: Creating...
module.rdsdb_sg.aws_security_group_rule.egress_rules[0]: Creating...
module.rdsdb.module.db_instance.aws_db_instance.this[0]: Creating...
module.public_bastion_sg.aws_security_group_rule.egress_rules[0]: Creation complete after 2s [id=sgrule-3684082686]
module.private_sg.aws_security_group_rule.egress_rules[0]: Creation complete after 2s [id=sgrule-3065790408]
module.loadbalancer_sg.aws_security_group_rule.ingress_with_cidr_blocks[0]: Creation complete after 2s [id=sgrule-14497190
module.rdsdb_sg.aws_security_group_rule.ingress_with_cidr_blocks[0]: Creation complete after 1s [id=sgrule-1115574092]
module.loadbalancer_sg.aws_security_group_rule.ingress_rules[0]: Creation complete after 3s [id=sgrule-3116469109]
module.rdsdb_sg.aws_security_group_rule.egress_rules[0]: Creation complete after 2s [id=sgrule-197316779]
module.private_sg.aws_security_group_rule.ingress_rules[1]: Creation complete after 3s [id=sgrule-3207872073]
module.vpc.aws_subnet.public[1]: Still creating... [10s elapsed]
module.vpc.aws_subnet.public[0]: Still creating... [10s elapsed]
module.vpc.aws_subnet.public[0]: Creation complete after 11s [id=subnet-04dae6b0d5f62f837]
module.vpc.aws_subnet.public[1]: Creation complete after 11s [id=subnet-09233acb5666c0101]
module.vpc.aws_route_table_association.public[1]: Creating...
module.vpc.aws_route_table_association.public[0]: Creating...
module.vpc.aws_nat_gateway.this[0]: Creating...
module.ec2_public.aws_instance.this[1]: Creating...
module.ec2_public.aws_instance.this[0]: Creating...
module.elb.module.elb.aws_elb.this[0]: Creating...
module.vpc.aws_route_table_association.public[1]: Creation complete after 1s [id=rtbassoc-0819627caab9067b3]
module.vpc.aws_route_table_association.public[0]: Creation complete after 1s [id=rtbassoc-0df574e2131a13d41]
module.rdsdb.module.db_instance.aws_db_instance.this[0]: Still creating... [10s elapsed]
```

Figure 4.7 Terraform Execution Timestamp



2023-05-02 18:15:18 UTC+0700	PublicRouteTable	CREATE_COMPLETE
2023-05-02 18:15:14 UTC+0700	LaunchTemplate	CREATE_IN_PROGRESS
2023-05-02 18:15:13 UTC+0700	InstanceSecurityGroup	CREATE_COMPLETE
2023-05-02 18:15:12 UTC+0700	InstanceSecurityGroup2	CREATE_COMPLETE
2023-05-02 18:15:12 UTC+0700	InstanceSecurityGroup	CREATE_IN_PROGRESS
2023-05-02 18:15:11 UTC+0700	InstanceSecurityGroup2	CREATE_IN_PROGRESS

Figure 4.8 Cloud Formation Stack Execution Timestamp

We measure each tools execution time 5 times and find the average for each tool. Please fine the result on the table 4.2 and Figure 4.10 below.

Table 4.2 Result of the execution Time Measurement

Resource	Terraform Execution Time (mm:ss)						CloudFormation Stack Execution Time (mm:ss)						
	T1	T2	T3	T4	T5	Avg.	T1	T2	T3	T4	T5	Avg.	Diff%
VPC Components	02:37	02:23	02:45	02:11	02:19	2:27	01:33	01:16	01:34	01:22	01:29	1:27	51.28
Security Group	00:26	00:24	00:30	00:26	00:25	0:26	00:02	00:02	00:02	00:02	00:02	00:02	171.43
Load Balance	00:06	00:06	00:04	00:06	00:06	0:06	00:02	00:02	00:02	00:02	00:02	00:02	100
EC2	02:14	02:16	02:09	02:24	02:14	2:15	01:11	01:19	01:07	01:14	01:13	01:13	58.9
Amazon RDS	17:50	18:44	18:23	17:55	18:09	18:00	14:40	14:47	14:35	14:22	14:06	14:29	21.65
Total	23:22	24:28	24:25	23:03	23:21	24:13	17:28	17:26	17:20	17:02	16:52	17:13	33.78

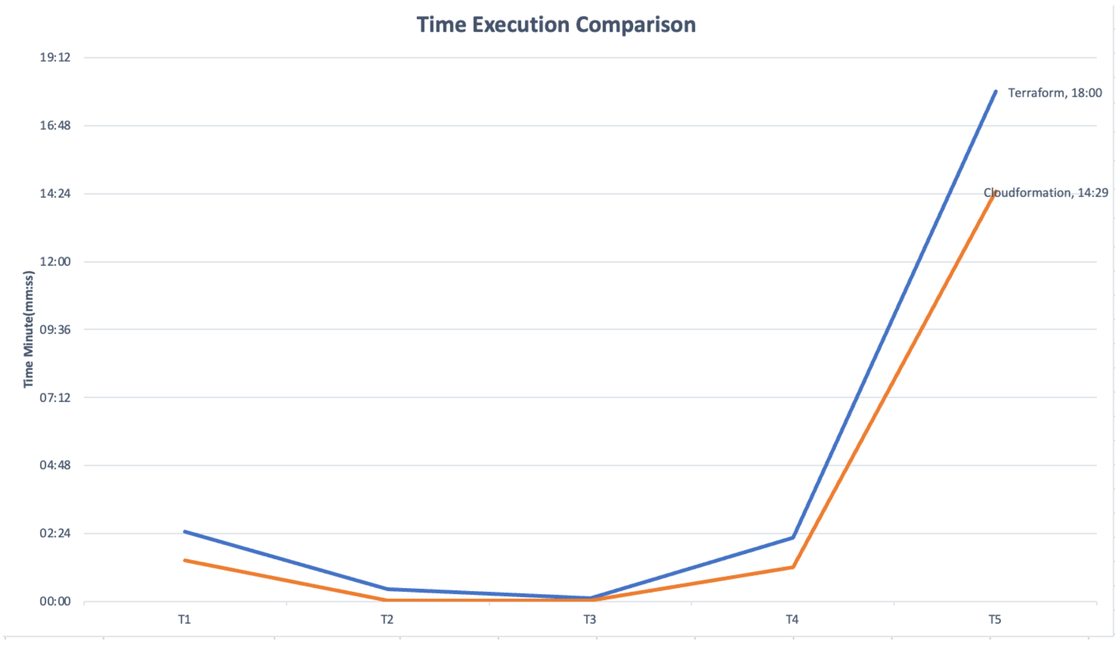


Figure 4.9 Time Execution Comparison

From Table 4.2, and Figure 4.8, We have found that Cloud Formation Stack Execution Time are 33.78% faster compared to terraform with almost 10 minutes different. The time for each task is also drastically different. For example, the big task like building the amazon RDS, Cloud Formation Stack is 4 min quicker. The build of 4 EC2 Instances, Cloud Formation Stack is 1 Min quicker as well and the build of VPC and its components, Cloud Formation is also 1 minute quicker. Overall, you can see that for the time execution standpoint, Cloud Formation Stack are much faster tool.

4.2.2 Device Resources Measurement Test Result (for Terraform Only)

To measure the devices' resource use during the terraform execution, we will measure by use the MacOS built-in activity monitoring tool. The data will be collecting every 5 Minutes during the execution then average out how much resource would be use.

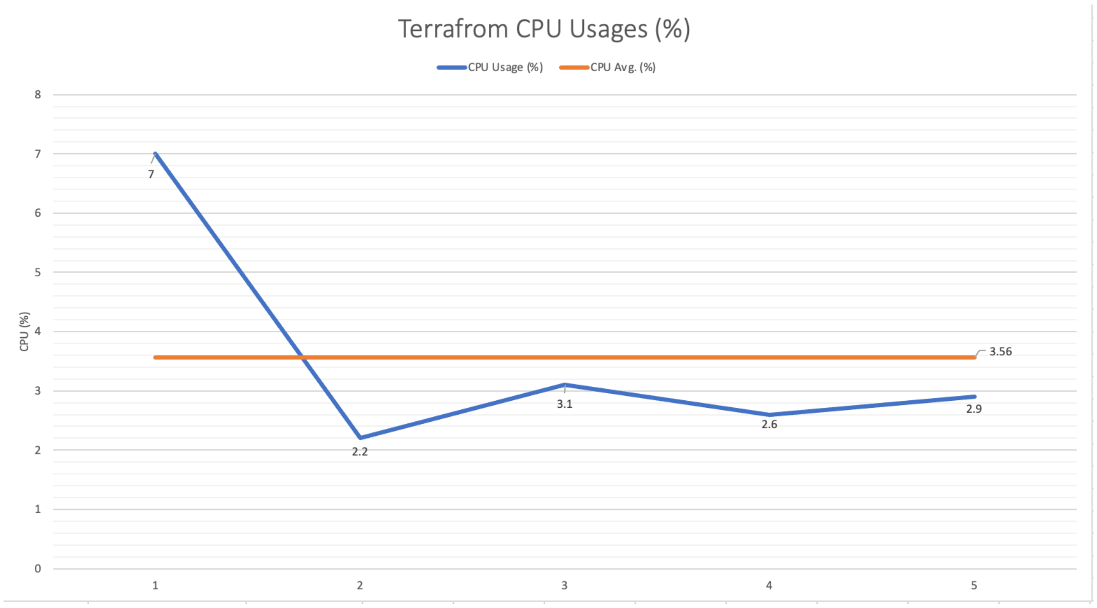


Figure 4.10 Terraform CPU Usage.

From Figure 4.9, the CPU usage for Terraform during the execution is not significant at all. It only averages out at 3.56%. Base on 10 Cores of CPU.

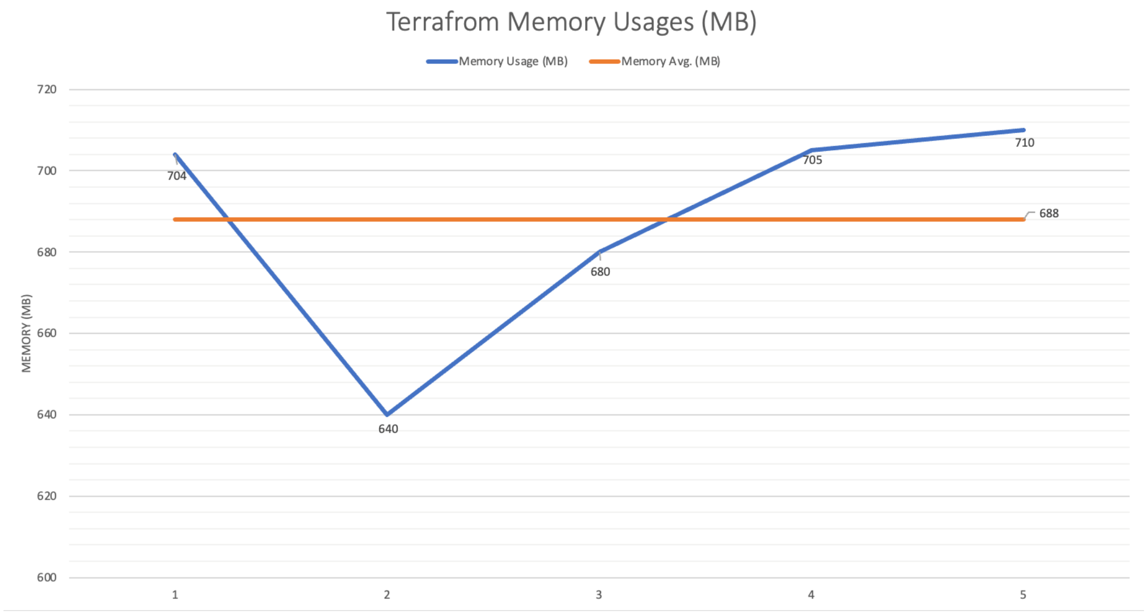


Figure 4.11 Terraform Memory Usage

From Figure 4.9, the memory usage for Terraform during the execution is averages out to 688.24MB. Which is not significant at all. The standard PC or MAC today should be able to handle the execution task easily.

4.3.3 Overall Efficiency Test Result, the overall result for efficiency will display table below.

Table 4.3 Efficiency Test Result

Test	Terraform	CloudFormation
Time Measurement Test	★★★	★★★★

Remark: ★= 30 Minutes up, ★★= 25-30 Minutes, ★★★= 20-25 Minutes, ★★★★= 15-20 Minutes, ★★★★★=1-15 Minutes

From Table 4.3, We can conclude that in terms of time efficiency, Cloud Formation stack has the edge. However, for the Resource Used efficiency we cannot conclude as we cannot measure Cloud Formation resource used as previously mentioned. Anyways, for terraform, we recommended that the spec of 4 Cores CPU and 4 GB of memory of a basic computer should be able to run Terraform without any issue.

4.3 Error Handling Test

On the Handling Test, we will test how each tools handling the error. The criteria that we will test are, how clear the error message display on the system? Are there any instruction on how to fix the error display? How to rerun after it fixed and how easy to find trouble shotting document online.

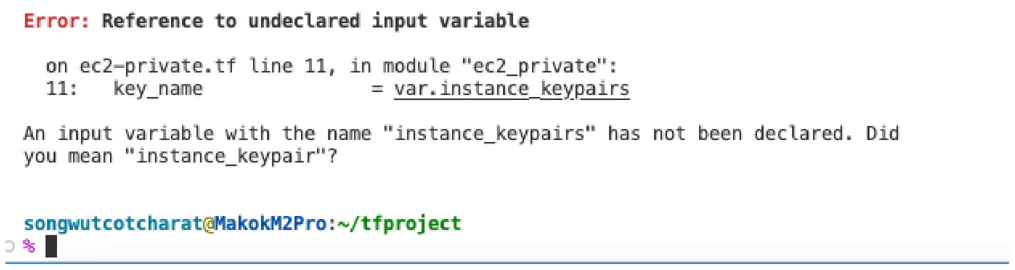
4.3.1 Terraform Error handling, Terraform has the Diagnostic Concept as follows.

4.3.1.1 Severity, Severity specifies whether the diagnostic is an error or a warning. Terraform, nor the framework, supports other severity levels. Use logging for debugging or informational purposes. An error will be displayed to the practitioner and halt Terraforms execution, not continuing to apply changes to later resources in the graph. We recommend using errors to inform practitioners about a situation the provider could not recover from. A warning will be displayed to the practitioner, but will not halt further execution, and is considered informative only. We recommend using warnings to inform practitioners about suboptimal situations that the practitioner should resolve to ensure stable functioning (e.g., deprecations) or to inform practitioners about possible unexpected behaviors.

4.3.1.2 Summary is a short, practitioner-oriented description of the problem. Good summaries are general—they don't contain specific details about values—and concise. For example, "Error creating resource", "Invalid value for foo", or "Field foo is deprecated".

4.3.1.3 Detail**,** Detail is a longer, more specific practitioner-oriented description of precisely what went wrong. Good details are specific—they tell the practitioner exactly what they need to fix and how. For example, "The API is currently unavailable, please try the request again.", "foo can only contain letters, numbers, and digits.", or "foo has been deprecated in favor of bar. Please update your configuration to use bar instead. foo will be removed in a future release".

4.3.1.4 Attribute, Attribute identifies the specific part of a configuration that caused the error or warning. Only diagnostics that pertain to a whole attribute or a specific attribute value will include this information.

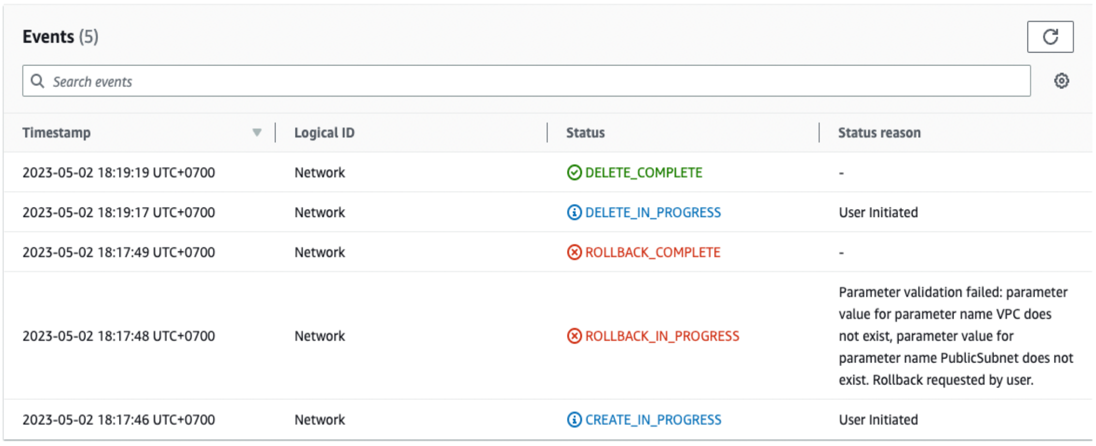


```
Error: Reference to undeclared input variable

  on ec2-private.tf line 11, in module "ec2_private":
  11:   key_name               = var.instance_keypairs

An input variable with the name "instance_keypairs" has not been declared. Did
you mean "instance_keypair"?

songwutcotcharat@MakokM2Pro:~/tfproject
% █
```

Figure 4.12 Example Terraform Error

From Figure 4.11 The Terraform error message is very clear, it tells you exactly what's wrong, which file is wrong, where is the line number causing the error. Also, it includes the recommendation on how you can fix it. The other positive point about the terraform error handling is, it will throw you the error since you do the command terraform plan. So, there will be no resources created if something is wrong with the configurations. Then you can go and fix the code before executing it again. The rerun process is after you fixed the code you can rerun the command from the top to execute everything again as previously showed on the capability test section.

4.3.2 CloudFormation Error handling, If AWS CloudFormation fails to create, update, or delete your stack, we can view error messages or logs to help us learn more about the issue. The following tasks describe general methods for troubleshooting a CloudFormation issue. Use the CloudFormation console to view the status of your stack. In the console, you can view a list of stack events while your stack is being created, updated, or deleted. From this list, find the failure event and then view the status reason for that event. The status reason might contain an error message from AWS CloudFormation or from a particular service that can help you troubleshoot your problem. For more information about viewing stack events. Aws website is also providing a significant amount of document on how you can handle the errors.



Events (5)

Timestamp	Logical ID	Status	Status reason
2023-05-02 18:19:19 UTC+0700	Network	DELETE_COMPLETE	-
2023-05-02 18:19:17 UTC+0700	Network	DELETE_IN_PROGRESS	User Initiated
2023-05-02 18:17:49 UTC+0700	Network	ROLLBACK_COMPLETE	-
2023-05-02 18:17:48 UTC+0700	Network	ROLLBACK_IN_PROGRESS	Parameter validation failed: parameter value for parameter name VPC does not exist, parameter value for parameter name PublicSubnet does not exist. Rollback requested by user.
2023-05-02 18:17:46 UTC+0700	Network	CREATE_IN_PROGRESS	User Initiated

Figure 4.13 Example Cloud Formation Stack Error.

From Figure 4.12 You can find the Terraform error message in a stack details menu somewhat clear, it tells you what’s is wrong, but does not give you the details on where to find the error. If the stack has failed it will stop the progress of that stack immediately. You have an option to fix the stack using the built-in designer code editor tool and rerun the stack or use your external coding IDE tool to fix the template and upload it again. Either way it very easy to rerun.

4.3.3 Error Handling Test Result, there are not much different on how easy or how hard the error handling for both tools, please see table 4.3 for the result details based on the criteria that mentioned previously.

Table 4.4 Error Handling Test Result

Test	Terraform	Cloud Formation
The error message	★★★★★	★★★★
Fix Instruction Message	★★★★★	★★★★
Rerun the failed task	★★★★★	★★★★★
Online instruction	★★★★★	★★★★★

Remark: The views and opinions expressed in this rating scale are those of the author and do not necessarily reflect the official rating or position of Terraform and Cloud

Formation. ★= Very Difficult to Handle, ★★= Difficult to Handle, ★★★= Not that Easy to Handle, ★★★★= Easy to Handle, ★★★★★= Very Easy to Handle

From Table 4.3, The result in terms of Error Handling Test, Terraform is a little bit better in the error message department as it showed much more clearer message even tell you which line of the code is having a problem. While the CloudFormation Stack does not give you that kind of details. On the other criteria, both online instructions can be found very easily.

4.4 Rollback Ability Test

On the Rollback ability test, every configuration management tools should have a roll back ability. The reason is in case there was a mistake on the creation of provisioning. The rollback ability gives you the chance to rollback to last working state. Both Terraform and Cloud Formation Stack have ability to rollback. The criteria that we will test are to see on which method of both tools are using to rollback and how complex is the rollback process would be.

4.4.1 Terraform Rollback Test, when execute the provisioning infrastructure resource using Terraform. Terraform will store the state of managed infrastructure and configuration. This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures.

This state is stored by default in a local file named "terraform.tfstate", butyou also have an option to store it on the terraform cloud to version, encrypt, and securely share it with your team.

Terraform uses state to determine which changes to make to your infrastructure prior to any operation, Terraform does a refresh to update the state with the real infrastructure.

The primary purpose of Terraform state is to store bindings between objects in a remote system and resource instances declared in your configuration. When Terraform creates a remote object in response to a change of configuration, it will record the identity of that remote object against a particular resource instance, and then potentially update or delete that object in response to future configuration changes.

```
backend = "local"

  config = {
    path = "..."
  }
}

# Terraform >= 0.12
resource "aws_instance" "foo" {
  # ...
  subnet_id = data.terraform_remote_state.vpc.outputs.subnet_id
}

# Terraform <= 0.11
resource "aws_instance" "foo" {
  # ...
  subnet_id = "${data.terraform_remote_state.vpc.subnet_id}"
}
```

Figure 4.14 Example Terraform State file

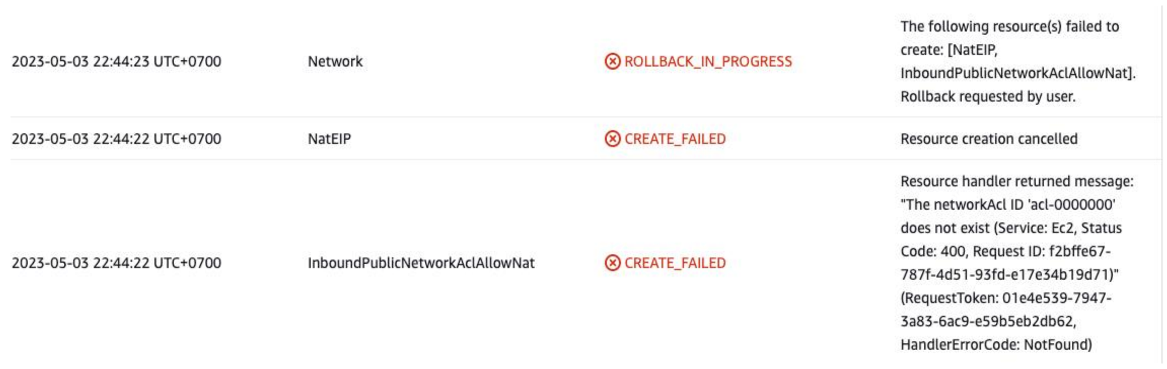
In order to rollback to the previous state of terraform. The command "terraform destroy" need to be run. This command will give you a chance to review on what would get destroy, you can review it before performing the rollback. After executing the command, the resources that we provisioned after the previous state file will get destroy and everything will revert to the previous state. When destroy finished it will show on how many resources has been destroy and how much time we are using to destroy each resource.

```
module.rdsdb.module.db_instance.aws_db_instance.this[0]: Destruction complete after
7m21s
module.rdsdb.module.db_parameter_group.aws_db_parameter_group.this[0]: Destruction
complete after 0s
module.rdsdb.module.db_subnet_group.aws_db_subnet_group.this[0]: Destruction
complete after 0s
module.rdsdb_sg.aws_security_group.this_name_prefix[0]:Destruction complete after 1s
module.vpc.aws_subnet.database[1]: Destruction complete after 1s
module.vpc.aws_subnet.database[0]: Destruction complete after 1s
module.vpc.aws_vpc.this[0]: Destruction complete after 1s
module.rdsdb.module.db_instance.aws_iam_role.enhanced_monitoring[0]: Destruction
complete after 3s
Destroy complete! Resources: 51 destroyed.
```

Figure 4.15 Example Terraform Destroy

4.4.2 Cloud Formation Stack Rollback Test, same with Terraform, Cloud formation has the also keep the state of your infrastructure that you can rollback to. Since Cloud Formation is a managed AWS service, it checks the infrastructure consistently to detect whether the provisioned infra is maintaining its state or not. CloudFormation receives a detailed response if anything changes. Cloud Formation stack has a few ways to rollback to previous state as follows.

4.4.2.1 Auto Rollback, If a stack fails to create by accident or user interaction, Cloud Formation executes a rollback meaning to delete all previously created resources. The stack itself stays in place in a rollback completed state to enable users to inspect and debug the problems. It is not possible to create this stack again. You need to delete the stack and then create the new template and create the stack again.

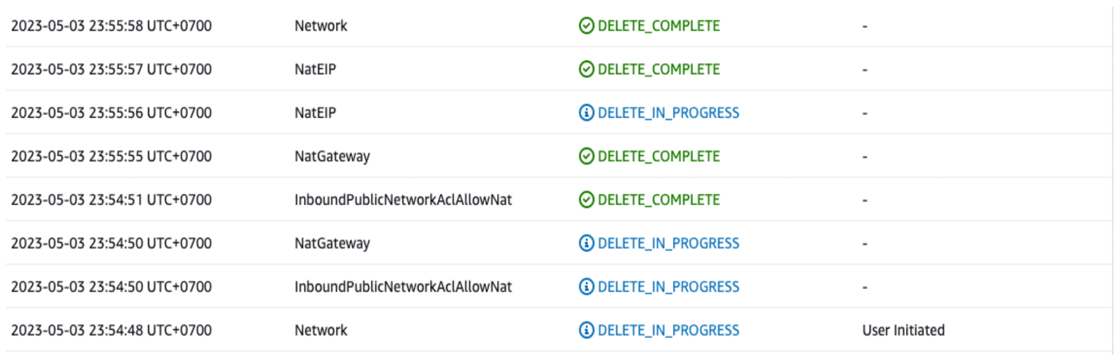


2023-05-03 22:44:23 UTC+0700	Network	⊗ ROLLBACK_IN_PROGRESS	The following resource(s) failed to create: [NatEIP, InboundPublicNetworkAclAllowNat]. Rollback requested by user.
2023-05-03 22:44:22 UTC+0700	NatEIP	⊗ CREATE_FAILED	Resource creation cancelled
2023-05-03 22:44:22 UTC+0700	InboundPublicNetworkAclAllowNat	⊗ CREATE_FAILED	Resource handler returned message: "The networkAcl ID 'acl-0000000' does not exist (Service: Ec2, Status Code: 400, Request ID: f2bffe67-787f-4d51-93fd-e17e34b19d71)" (RequestToken: 01e4e539-7947-3a83-6ac9-e59b5eb2db62, HandlerErrorCode: NotFound)

Figure 4.16 Example Cloud Formation Stack Auto Rollback.

From Figure 4.12, when the stack has reached failed state. The auto rollback will kick in immediately to roll back everything to the previous state.

4.4.2.2 Manual Rollback, If the stack complete successfully and resources have been created on AWS already. The only way to roll back to the previous state is to delete the stack that already create. Then all the resources that have been provisioned will roll back to the previous state.



2023-05-03 23:55:58 UTC+0700	Network	✓ DELETE_COMPLETE	-
2023-05-03 23:55:57 UTC+0700	NatEIP	✓ DELETE_COMPLETE	-
2023-05-03 23:55:56 UTC+0700	NatEIP	ⓘ DELETE_IN_PROGRESS	-
2023-05-03 23:55:55 UTC+0700	NatGateway	✓ DELETE_COMPLETE	-
2023-05-03 23:54:51 UTC+0700	InboundPublicNetworkAclAllowNat	✓ DELETE_COMPLETE	-
2023-05-03 23:54:50 UTC+0700	NatGateway	ⓘ DELETE_IN_PROGRESS	-
2023-05-03 23:54:50 UTC+0700	InboundPublicNetworkAclAllowNat	ⓘ DELETE_IN_PROGRESS	-
2023-05-03 23:54:48 UTC+0700	Network	ⓘ DELETE_IN_PROGRESS	User Initiated

Figure 4.17 Manual Delete for Rollback Cloud Formation Stacks

From Figure 4.13, if all the resource shas been create and provisioned already, User need to initiate the delete of Cloud Formation Stack in order to manually rollback to previous state.

4.4.3 Roll back Test Result

Table 4.5 Roll back Test Result

Test	Terraform	Cloud Formation
State Lock (State file to roll back)	★★★★★	★★★
Rollback Method	★★★★	★★★★★
Difficulty	★★★★★	★★★★★
Online instruction	★★★★★	★★★★★

Remark: The views and opinions expressed in this rating scale are those of the author and do not necessarily reflect the official rating or documentation of Terraform and Cloud Formation. ★= Very Difficult to Handle, ★★= Difficult to Handle, ★★★= Not that Easy to Handle, ★★★★= Easy to Handle, ★★★★★= Very Easy to Handle.

From Table 4.3, The result in terms of Rollback Ability Test, Both tools have a store state file. The rollback method is a bit better on Cloud Formation since it has the auto rollback. Both are easy to roll back and have a robust online instruction.

CHAPTER 5
RESULT SUMMARY AND SUGGESTIONS

In This chapter we'll bring all the test result to conclusion on what would be the recommendation Infrastructure as code tool to choose. The multiple tests were already done based on five decision criteria. The test was also done multiple times to make sure that we get the most accurate result as much as possible. It will also include the discussion point on what should be improve that related to the objective of the research and recommendations point on these Infrastructure as Code tool that will change the world of how we provision the resources and infrastructure on cloud.

5.1 Result Summary

5.1.1 Capability Test Result Summary

Based the criteria that all the infrastructure resource types that needs to be provisioned on AWS to accommodate the used of 3 tier application. All are them were successfully built using both Terraform and AWS CloudFormation Stack. There might be some differences on how each function and service was build. But overall, the result is very satisfied that cannot concluded on which tools is better in terms of their capabilities.

5.1.2 Efficiency Test Result Summary

From the efficiency test standpoint, we can only measure how much time each tools are using for execute the same provision resources tasks on AWS. We also can only measure the device's CPU/Memory used during the terraform execution only, since CloudFormation does not require a software to execute the task. It will execute directly on AWS GUI console. From the time measurement test, we have found that, Overall execution time of CloudFormation are 33.78% or 10 Minutes faster compared to terraform. Which is very significant different.

The conclusion here is that terraform is running on another environment or system and that it will send out the configuration data to AWS. On the other hands, Cloud Formation Stack is executed direct on AWS as it is one of their manage services, so the execution is running on AWS resource that's why the running time are much faster using Cloud Formation.

5.1.3 Error Handling Test Summary

The result of the handling test is that Terraform has much more clearer error message compared to CloudFormation Stack. The Terraform message even guide you on how and where to fix the issue. On the other hands, CloudFormation Stack give you enough information for you to work on the error but sometimes don’t tell you exactly where to go and look for the issue. The fix for the error for both tools are quite easy, if the problem is found and need to be fix, generally go to the code change whatever it need to be change, then re-run them again. The other thing that needs to point out is that, both tools have a significant amount of online instruction and trouble shooting. Enough for anyone to read and troubleshooting by tier own. The conclusion here is that Terraform might have an edge, because they are providing much clearer message for debugging.

5.1.4 Rollback Ability Test Summary

The result of the roll back ability test came out that, both tools have the state file that will keep the previous state of infrastructure configuration or provisioning. So, it quite useful and easy for the roll back. The difference is the state file for Terraform can be store either on a local for a single user or a remote location that can be access by the team. While CloudFormation Stack state file is fully managed by AWS, as it makes a lot of sense since the Cloud Formation is the AWS managed service. On the rollback steps the Cloud Formation have the auto-roll back maneuver when the stack is failed, which is very useful. On the other hands Terraform have only manual rollback maneuver. So, we can conclude that Cloud Formation Stack are better in this department because of the auto-roll back maneuver.

5.1.5 Physical Differences and Limitations Test Conclusion

The conclusion of this might not be conclusive enough, since it more of the information on test are just from the observation of the researcher that encounter along the way. Anyways, there are many points that the researcher can point out here such as Cloud Formation Stack only support AWS. While Terraform support many platforms or the language that both tools used are drastically different. So, Engineer needs to choose carefully on which tools is better with their skill set. Another limitation is both tools are mostly made for provisioning infrastructure. It cannot use for the configuration management or application deployment on their own. As there are many other tools that can help with those tasks.

5.2 Overall Conclusion

Table 5.1 Overall Conclusion

Criteria	Terraform	Cloud Formation
Capability	✓	✓
Efficiency	3	4
Error Handling	20	18
Rollback Ability	19	18

Remark: The number presented here is the accumulate of the star rating that the author gave out on Chapter four. The views and opinions expressed in this rating scale are those of the author and do not necessarily reflect the official rating or documentation of Terraform and CloudFormation.

✓ = Successful ✗ = Unsuccessful

From Table 5.1, According to the numbers, Cloud Formation is better in terms of time efficiency. Terraform is better in rollback ability and error handling. For capability both tools are measured to be equally good as it successful to run every tasks.

As we have seen, both Cloud Formation and Terraform offer powerful IaC capabilities, but it is important to consider your workload, team composition, and infrastructure needs when selecting IaC platform.

Cloud Formation is a better option if your entire infrastructure is on AWS and there are no plans to go multi-cloud. If you are new to AWS services, native support would be beneficial. It is built by AWS and has faster AWS-related updates. It also uses JSON and YAML, so there is no learning curve as opposed to HCL.

Terraform is the best option if you are using or planning to use multi-cloud resources. The modular approach allows you to create reusable templates, which speed up the configuration process.

The bottom line is this research still cannot be undetermined on what is best for you, or your organization is depends on the requirements. The researcher recommends selecting the IaC tools after evaluating your application's infrastructure strategy.

5.3 Suggestions

5.3.1 Try to create more sophisticated infrastructure than 3 tier web application and see how both tools response.

5.3.2 Use both tools to complement each other’s when building the infrastructure and see how they fit in with each other’s.

5.3.3 Not just use both tools to build or provisioning the infrastructure but try to use in the configuration management also. For example, try to deploy application using the tools and see how they are response.

BIBLIOGRAPHY

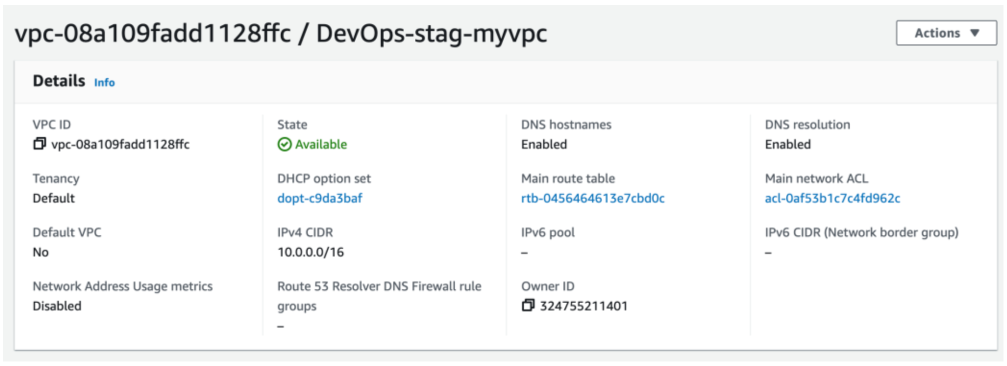
BIBLIOGRAPHY

1. Artac, M., Borovssak, T., Di Nitto, E., Guerriero, M., & Tamburri, D. A. (2017). DevOps: Introducing Infrastructure-as-Code. In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), (pp. 497-498).
2. Brikman, Y. (2019). Terraform: Up & Running: Writing Infrastructure as Code. O'Reilly Media, Inc.
3. Campbell, B. (2020). The Definitive Guide to AWS Infrastructure Automation: Craft Infrastructure-as- Code Solutions. Apress.
4. Guerriero, M., Garriga, M., Tamburri, D. A., & Palomba, F. (2019). Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 580-589).
5. Labouardy, M. (2021). Pipeline as Code: Continuous Delivery with Jenkins, Kubernetes, and Terraform. Manning Publications.
6. Alexander, S. (2019). [Online]. https://www.techtarget.com/searchioperations/-definition/Terraform.
7. Contino (2022). [Online]. https://www.contino.io/insights/aws-cloudformation.
8. zeroFruit (2022). Hashicorp Plugin System Design and Implementation by zeroFruit Medium.
9. Murphy, O. (2022). Adoption of Infrastructure as Code (IaC) in Real World; Lessons and practices from industry. JAMK University of Applied Sciences.
10. Bloom, D. (2017). [Online]. https://www.thegreengrid.org/en/newsroom/blog/software-development-discipline-reshapes-infrastructure.
11. Taylor, T. (2022). [Online]. https://amazic.com/top-3-open-source-tools-that-enable-infrastructure-as-code/.
12. Ramamoorthy, C. (2021). [Online]. https://www.linkedin.com/pulse/testing-most-critical-cog- your-devops-wheel-chandrasekar-ramamoorthy.
13. Mike (2022). [Online]. https://www.morethansap.com/2022/01/24/devops-infrastructure-as-code-and-sap/.
14. Stella, J. (2021). [Online]. https://www.infoworld.com/article/3634774/how-to-secure-cloud-infrastructure-across-the-development-lifecycle.html.

APPENDIX

APPENDIX A
Result on AWS Console

VPC result



vpc-08a109fadd1128ffc / DevOps-stag-myvpc

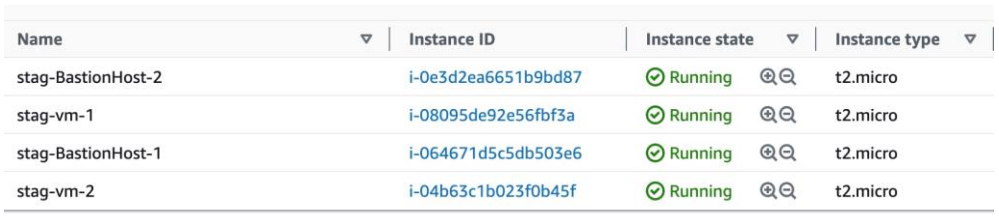
Actions ▼

Details Info

VPC ID vpc-08a109fadd1128ffc	State ✓ Available	DNS hostnames Enabled	DNS resolution Enabled
Tenancy Default	DHCP option set dopt-c9da3baf	Main route table rtb-0456464613e7cbd0c	Main network ACL acl-0af53b1c7c4fd962c
Default VPC No	IPv4 CIDR 10.0.0.0/16	IPv6 pool –	IPv6 CIDR (Network border group) –
Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups –	Owner ID 324755211401	

Figure A.1 VPC created on AWS

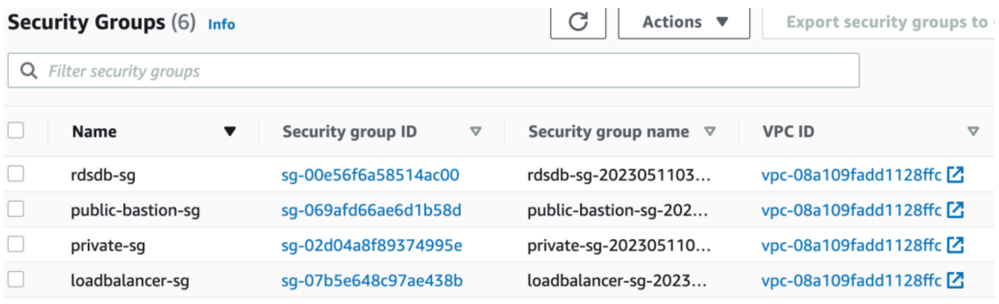
EC2 Result



Name	Instance ID	Instance state	Instance type
stag-BastionHost-2	i-0e3d2ea6651b9bd87	✓ Running	t2.micro
stag-vm-1	i-08095de92e56fbf3a	✓ Running	t2.micro
stag-BastionHost-1	i-064671d5c5db503e6	✓ Running	t2.micro
stag-vm-2	i-04b63c1b023f0b45f	✓ Running	t2.micro

Figure A.2 Private and Public EC2 instances has been created

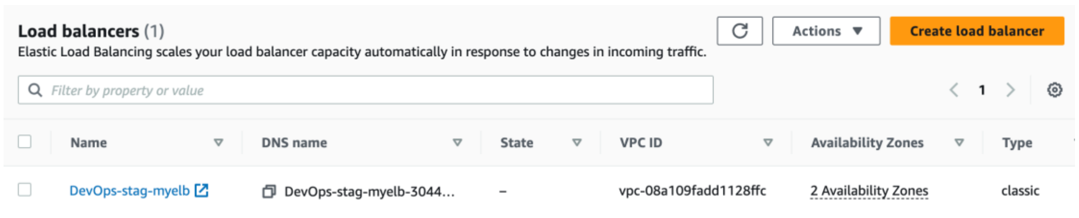
Security Group Result



Name	Security group ID	Security group name	VPC ID
rdsdb-sg	sg-00e56f6a58514ac00	rdsdb-sg-2023051103…	vpc-08a109fadd1128ffc
public-bastion-sg	sg-069afd66ae6d1b58d	public-bastion-sg-202…	vpc-08a109fadd1128ffc
private-sg	sg-02d04a8f89374995e	private-sg-202305110…	vpc-08a109fadd1128ffc
loadbalancer-sg	sg-07b5e648c97ae438b	loadbalancer-sg-2023…	vpc-08a109fadd1128ffc

Figure A.3 Security Group has been created for each resource

Load Balancer Result

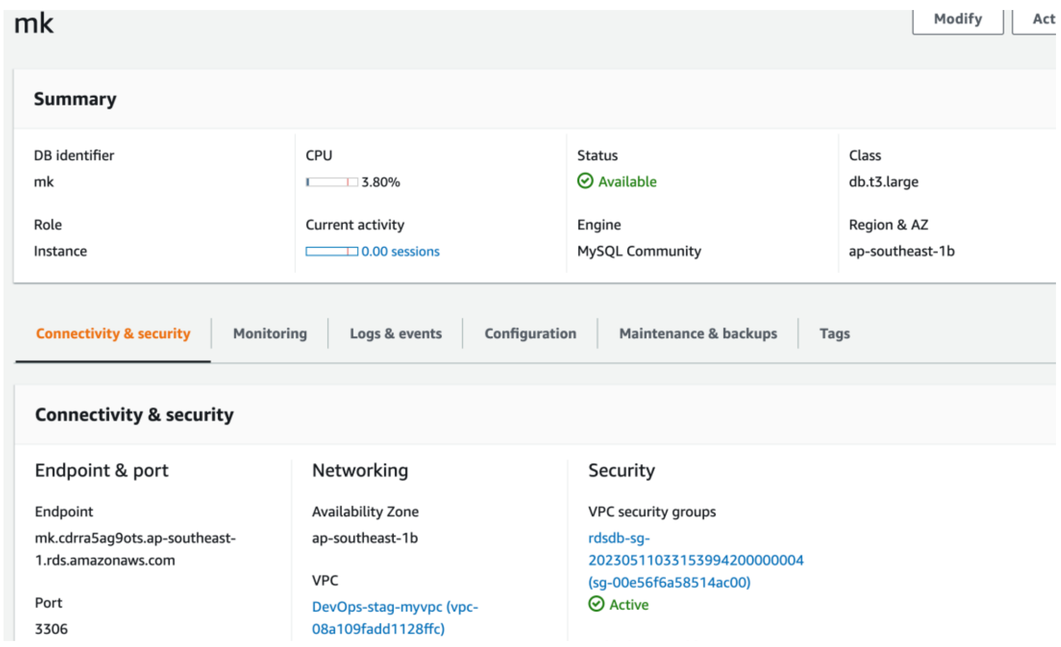


Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

Name	DNS name	State	VPC ID	Availability Zones	Type
DevOps-stag-myelb	DevOps-stag-myelb-3044...	–	vpc-08a109fadd1128ffc	2 Availability Zones	classic

Figure A.4 Load Balancer has been created

RDS Result



DB identifier mk	CPU 3.80%	Status Available	Class db.t3.large
Role Instance	Current activity 0.00 sessions	Engine MySQL Community	Region & AZ ap-southeast-1b

Connectivity & security

Endpoint & port
Endpoint: mk.cdrra5ag9ots.ap-southeast-1.rds.amazonaws.com
Port: 3306

Networking
Availability Zone: ap-southeast-1b
VPC: DevOps-stag-myvpc (vpc-08a109fadd1128ffc)

Security
VPC security groups: rdsdb-sg-20230511033153994200000004 (sg-00e56f6a58514ac00) Active

Figure A.5 RDS Instance has been created

CURRICULUM VITAE

Name	Songwut Cotcharat

Education

ค.ศ. 2004	- BBA, Business Computer, Bangkok University.

Work Experience

ค.ศ. 2023	- Present, Cloud SRE, Zilo Asia.
ค.ศ. 2022	- IT Operations Manager, Fujitsu Thailand.
ค.ศ. 2019	- Head of Technical Support, ThisFish Inc.
ค.ศ. 2011	- IT Operations Manager, SS&C Technologies.