A Comparative Study on the Provisioning Time of Terraform and Ansible in Infrastructure as Code Deployment

by

Divya Maria Appachan

South East Technological University
School of Science and Computing
MSc in Computing (Enterprise Software Systems)

Supervised by
Jimmy Mcgibney

December 17, 2023

Abstract

This research paper presents a comprehensive comparative study of two popular infrastructure as code solutions, Terraform and Ansible. Infrastructure as Code has become a fundamental practice in modern IT operations, enabling the automation and management of cloud infrastructure and resources. Terraform and Ansible are two leading infrastructure as Code tools that have gained prominence in the field. The research methodology involves evaluating the provisioning speed of Terraform and Ansible.

Keywords: Infrastructure as Code, Terraform, Ansible, Automation, Cloud Infrastructure, Comparative Study,provisioning speed.

Contents

List of Figures

1 Introduction

A fundamental component of the DevOps culture, infrastructure as code attempts to bridge the gap between development and operations. Software releases used to be a difficult, risky, and time-consuming procedure before DevOps[1]. With the aid of DevOps, the software release difficulty has now been reduced, ensuring the developers to focus on the process of developing the product. Components from physical data centers, such as virtual machines (VMs), networks, load balancers, and numerous other environments for hosting applications, can be created by code with the aid of infrastructure as code technologies. DevOps is able to swiftly provision, deprovision, and expand infrastructure in response to changing requirements once it has been coded. The development, testing, and deployment of software have been substantially accelerated by this agility.

The key benefits of using infrastructure as code are that it drastically cuts down on time-to-production, enhances consistency, shields against "configuration drift" issues, reduces costs, and preserves expertise inside enterprises [2]. Benefits should be examined while selecting infrastructure as a code tool for a certain project. Terraform and Anisable are two highly powerful open-source IT solutions that are extensively employed in industry. Ansible, a command-line software program for IT automation that is open source, provides a more extensive configuration management history and was initially released in 2012[3]. Terraform is considered an orchestration tool by HashiCorp that uses the declarative language HashiCorp Configuration Language (HCL) or alternatively JavaScript Object Notation (JSON) [4], which was made available in 2014. Infrastructure deployment and termination are supported by both platforms.

.ft Terraform file for building a security group, key pair and Amazon Elastic Compute Cloud (EC2) instance with the t2.nano instance type:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region     = "AWS_REGION"
  access_key = "AWS_ACCESS_KEY"
  secret_key = "AWS_SECRET_KEY"
}

// To Generate Private Key
resource "tls_private_key" "tls_private_key" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

variable "key_name" {
  description = "Name of the SSH key pair"
}

// Create Key Pair for Connecting EC2 via SSH
resource "aws_key_pair" "key_pair" {
  key_name   = var.key_name
  public_key = tls_private_key.tls_private_key.public_key_openssh
}

// Save PEM file locally
resource "local_file" "private_key" {
  content  = tls_private_key.rsa_4096.private_key_pem
  filename = var.key_name
}
```

```
# Create a security group
resource "aws_security_group" "SG" {
  name        = "SG"
  description = "Security group for EC2"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "public_instance" {
  ami                    = "ami-0f5ee92e2d63afc18"
  instance_type          = "t2.nano"
  key_name               = aws_key_pair.key_pair.key_name
  vpc_security_group_ids = [aws_security_group.SG.id]

  tags = {
    Name = "public_instance"
  }

  root_block_device
```

Ansible is procedural methodology playbooks written in Yet Another Markup Language (YAML) shown below. However, some contend that Ansible can offer orchestration. This study attempts to assess the performance of Terraform and Ansible by obtaining empirical provisioning timing of code.
.yaml Ansible file for building a security group, key pair and Amazon Elastic Compute Cloud (EC2) instance with the t2.nano instance type:

```
- hosts: localhost
  gather_facts: false

  tasks:
    - name: Generate RSA Private Key
      tls_private_key:
        path: "my-key-name"
        type: rsa
        size: 4096

    - name: Create Security Group
      ec2_group:
        name: SG
        description: Security group for EC2
        ingress_rules:
          - proto: tcp
            from_port: 22
            to_port: 22
            cidr_ip: 0.0.0.0/0
        egress_rules:
          - proto: -1
            cidr_ip: 0.0.0.0/0
      register: security_group_info

    - set_fact:
        security_group_id: "{{ security_group_info.group_id }}"

    - name: Provision EC2 Instance
      ec2_instance:
        key_name: "my-key-name"
        instance_type: "t2.nano"
        image_id: "ami-0f5ee92e2d63afc18"
        security_group: "{{ security_group_id }}"
        instance_tags:
          - key: Name
            value: public_instance
        root_volume_size: 5
        root_volume_type: gp2
```

2 Research Questions

To accelerate the software development life cycle, infrastructure as code is an important practice. In this research on infrastructure as code, the following questions are answered:

1. Research Question 1: Which infrastructure as code tool is more efficient in terms of provisioning speed when creating a cloud infrastructure?

   The commands "terraform apply" and "terraform destroy" for Terraform are used to set up and delete the infrastructure code on the requested platform. On the other hand, actions involving servers and infrastructure resources are configured and automated using Ansible commands such as "ansible-playbook." One crucial element is the time required to finish creating the whole infrastructure in the code after execution of those commands. Reducing the amount of time spent provisioning helps achieve zero downtime while releasing new application and updating infrastructure. Conversely, financial, health, and insurance-related projects focus a strong emphasis on Service Level Indicator (SLI), Service Level Objective (SLO), and Service Level Agreement (SLA). These metrics may be affected by configuration changes.

2. Research Question 2: Which tool is better suited to a particular task when compared to the alternative option?

   Certain software projects require huge volumes of data to be stored and processed, which means that numerous tables, identity and access management(IAM) policies, storage buckets, and security keys must be created on a daily basis on cloud platform. In order to choose the best infrastructure as Code tool for a project involving this kind, it is helpful to assess the Infrastructure as Code tool efficiency in terms of productivity speed based on the work involved in a particular task

3. Research Question 3: Does the provisioning speed for specific infrastructure as code tools depend on the unique features selected for specific cloud deployments, or are specific tools just more efficient than others?

   In order to optimize their infrastructure as code workflows, practitioners must choose an infrastructure as code solution that takes into account the differences in provisioning speeds between various cloud resources. For example EC2 instances on the AWS platform, exploring diverse specifications. This encompasses variations in boot volume types, regions, availability zones, tenancy models, and instance types and sizes. Observe the provisioning time and its influence on the creation of resources with varied specifications. The answers to this research question can offer details that will be useful in decision-making when choosing both infrastructure as code tools in accordance with cloud service providers.

3 Methodology

The research focuses on statistical analysis through experimentation to propose the most effective instrument for evaluating the provisioning speed of cloud infrastructure. Using Terraform and Ansible on cloud provider, the exact identical infrastructure was created for this experiment. Analyse the provisioning time while creating, updating, and destroying cloud infrastructure resources such as virtual machines, load balancers and cloud storage hosted on Amazon Web Services(AWS).
Based on empirical evidence and analysis determine which tool is suited for specific task such as
-Parallel task
-Configuration task
Creating numerous IAM roles, storage buckets, and security groups are illustrations for parallel tasks. The configuration job may be tested by building a virtual machine, running different web servers, such as Nginx and Apache HTTP Server and installing different software packages.

3.1 Sample Architectural Configuration Overview

The following is an overview of the sample architecture for this research during the implementation stage, using Terraform and Ansible.

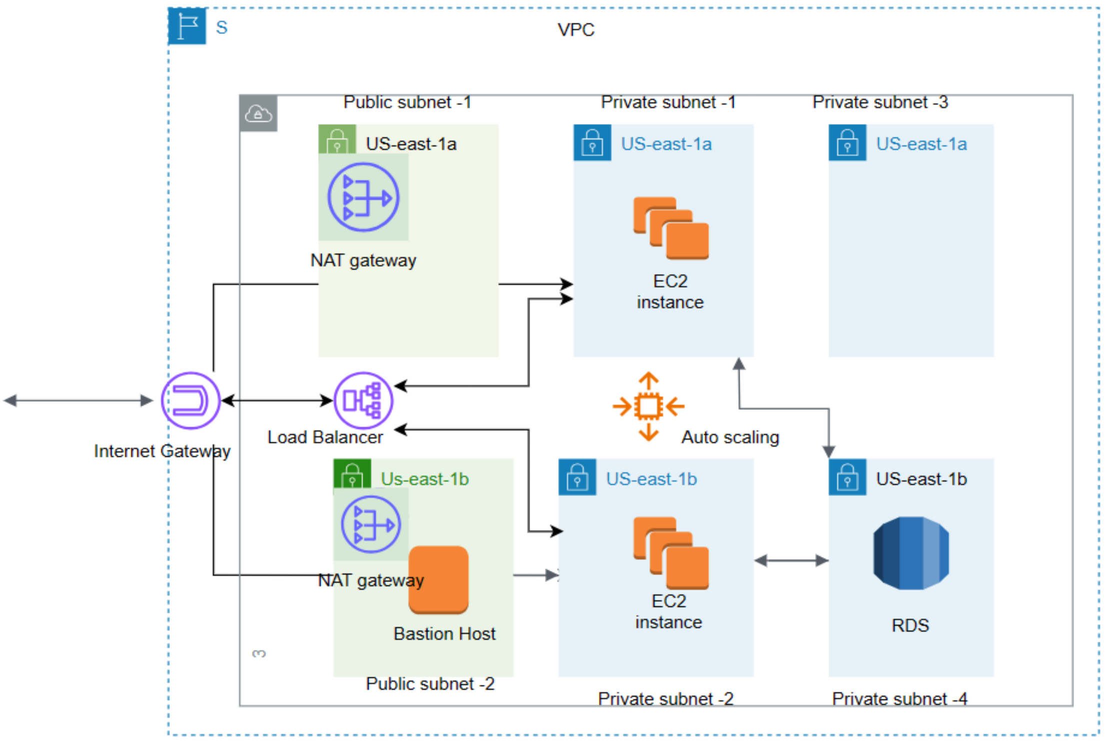

Figure 1: First Sample architecture

The diagram above depicts the deployment of a web application that includes a 3-tier virtual private cloud(VPC), load balancing, and auto-scaling.VPC with two availability zones with public and private subnets. In order to facilitate communication between VPC instances and the internet, an internet gateway is utilised. Public subnets are used by resources like application load balancers, bastion hosts, and Network address translation(NAT) gateways. To safeguard the web servers and database servers, we will place them in private subnets. The instances in the private data and app subnets may access the internet thanks to the NAT gateway. The website is hosted on EC2 instances and database is a MySQL RDS. The web traffic is distributed among an auto-scaling group of EC2 machines in several availability zones using an application load balancer. To address the third research question, deploy the architecture with different specifications of the auto-scaling instances in the upcoming semester and evaluation of provisioning time differences.

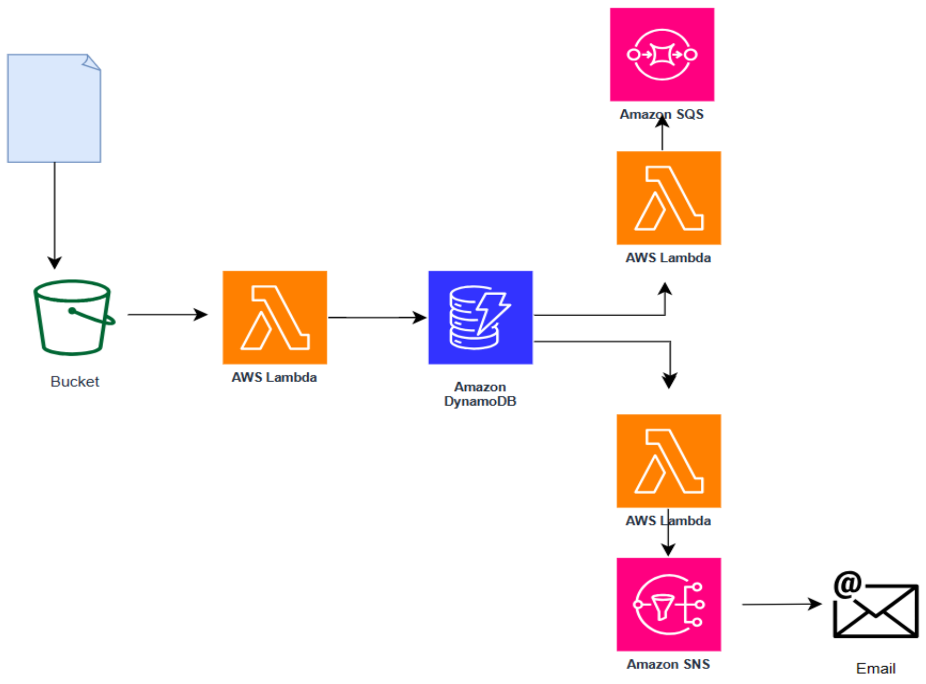

Figure 2: Second Sample architecture

The system design is a serverless system hosted on AWS. Uploading files to an Amazon Simple Storage Service(Amazon S3) bucket allows Lambda functions to process and update an Amazon DynamoDB database. Notifications are sent using Amazon Simple Notification Service (SNS), when there is change in Dynamodb. Another Lambda function used to retrieve change via Simple Queue Service (SQS) is also linked to a DynamoDB table. The cloud architecture diagram depicts the essential cloud services and components.

A couple more Scenarios will be constructed using Terraform and Ansible, and the time will be recorded.Also, try deploying the same architecture with a different resource specification. Consider the first example: the architecture will replicate with a new EC2 machine type and availability zone and measure the provisioning time.

3.2 Terraform

Terraform is an infrastructure as code tool developed and maintained by HashiCorp help building and managing a specific infrastructure in the configuration files. Using its application programming interfaces (APIs), Terraform develops resources on cloud platforms and other services. Terraform uses state to decide which infrastructure modifications to apply. When we create infrastructure in the Terraform setup, a state file called terraform.tfstate is created; state files must be stored securely to ensure the infrastructure continues to work. We can define resources that span many, allowing configurations like deploying applications on VM in a VPC with a load balancer and security groups. With the ”terraform plan” command, Terraform then creates an execution plan with specific infrastructure tasks based on the settings that are already in place. When these activities are approved, ”terraform apply” them in a predetermined order to guarantee that resource dependencies are met. For example, a VPC is recreated before VM are scaled in response to changes and the state file is updated. Similarly, after upgrading the code and plan, user can use the ”terraform destroy” command to decommission the infrastructure. The Figure 3 shows the workflow of the Terraform.

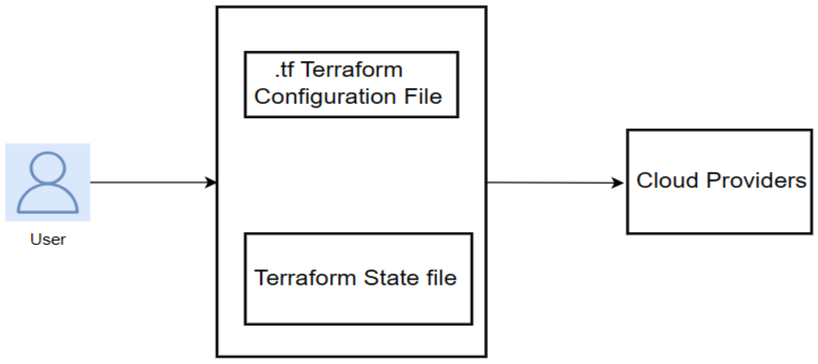

Figure 3: Terraform Workflow

3.3 Ansible

Ansible is a configuration management, orchestration, and deployment agentless automation tool that is becoming more and more well-liked due to its ease of use, effectiveness, and absence of target machine software requirements. Ansible Server, which connects to target nodes via password-free SSH, is a component of its architecture. Playbooks are YAML-formatted Ansible code files. The playbook can then be used to send new configurations or confirm the configuration of remote systems. The hosts and groups of hosts on which commands, modules, and tasks in a playbook perform are defined in the Ansible inventory file. The flow includes creating playbooks and inventories locally, establishing SSH connections, information about target nodes is gathered, and playbooks are run on the nodes for optimised automation. The Figure 4 shows the workflow of the Terraform.

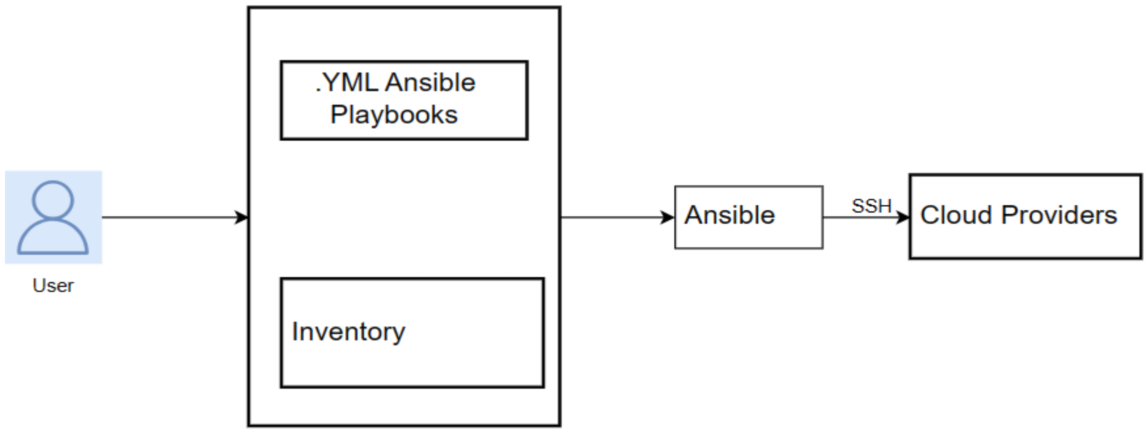

Figure 4: Ansible Workflow

4 Literature Review

A wide array of literature is available on the concept of Infrastructure as Code within the software development lifecycle and its pivotal role in DevOps. Infrastructure as code empowers IT to manage infrastructure responsibilities encompassing provisioning, deployment, configuration, and orchestration with the aid of diverse Infrastructure as Code solutions currently accessible.

Ankita Dalvi[5] provides helpful insights about the challenges associated with cloud adoption. The focus is on the important role that Cloud Centre of Excellence (CCoE), or centralised platform teams, play in simplifying application delivery in multi-cloud systems. The paper acknowledges the challenges of manual provisioning, which is time-consuming processes and chance for errors, and offers a new approach designed to provide developers more control. Infrastructure as code templates are the base of the suggested approach, which provide self-service for on-demand platform capabilities access. The paper highlight the development of cloud architectures and some important results from HashiCorp's 2021 State of the cloud survey report in discussion of the use of multiple clouds. The paper provides a detailed description of how cloud infrastructure is managed by centralised teams, highlighting the importance of strong security and management. It describes the difficulties platform teams have in delivering uniform procedures across different development teams and makes possible for the use of infrastructure as code to identify and codify repetitive trends. The proposed model is compared with the present model, which separates template development and provisioning and is characterised by operational delays and human interventions. The proposed model promises increased efficiency and reduced dependence on platform teams. In the end, the paper describes an effective architecture, including all of its components and benefits such as avoiding delays, providing distribution control, and improving visibility and governance. The system's deployment results show how effective it is in considerably reducing provisioning time when compared to previous methods. Together with future development options, the study also describes the overall effects of the proposed system on agility and cloud resource management.

Michael Howard [6] makes pertinent observations regarding the significance of rigor in software development and the challenges associated with manual environment provisioning. In practice, the introduction of Infrastructure-as-Code solutions like Terraform has revolutionized modern IT processes. By representing infrastructure as code, organizations attain uniformity, speed, and reliability in their infrastructure implementation and management. Terraform, renowned for its adaptability, user-friendliness, and compatibility with a multitude of cloud and infrastructure providers, stands as a preferred choice. The deployment of infrastructure through Terraform involves three fundamental steps: code, plan, and apply, enhancing organizations' ability to construct a more robust and efficient software delivery pipeline.

Ansible, a widely utilized Infrastructure as Code tool akin to Terraform, takes the spotlight in the paper titled "Applications in Amazon Web Services Using Ansible, Jenkins, and Kubernetes" [7]. This paper primarily illustrates an automated DevOps pipeline for the deployment of Java-based web applications on AWS. The authors emphasize the significance of cloud computing, including service models such as IaaS, PaaS, and SaaS. Furthermore, the paper delves into the role of DevOps practices and the critical role of automation within Continuous Integration and Continuous Deployment (CI/CD) processes, with special focus on Ansible. It introduces various CI/CD tools, with Ansible occupying a central role. A concise comparison of these tools is also provided. The research's core objective is to showcase an automated pipeline that facilitates the efficient deployment of applications on AWS while adhering to DevOps best practices, with Ansible serving as a cornerstone for Continuous Deployment. It's important to note that several alternative Infrastructure as Code tools, including AWS CloudFormation, Azure Resource Manager, Chef, Puppet, and Saltstack to perform infrastructure task.

The study titled "On the Effectiveness of Tools to Support Infrastructure as Code: Model-Driven Versus Code-Centric" [8] undertakes a comparative analysis, pitting a model-driven tool called Argon against the popular code-centric tool Ansible for defining cloud infrastructure. The study's primary objective is to furnish empirical evidence of the effectiveness of these tools and to gauge user perceptions. Three comprehensive experiments, involving 67 Computer Science students, were conducted to assess parameters including effectiveness, efficiency, convenience of use, perceived utility, and the intention to use these tools. Employing the AB/BA crossover design and employing the linear mixed model for statistical analysis, the results and ensuing meta-analysis demonstrate that Argon outperforms Ansible in supporting infrastructure as code for defining cloud infrastructure. Participants found Argon more user-friendly and useful for specifying infrastructure resources. The results of this study indicate that Argon expedites the provisioning process by automating script generation for a range of DevOps tools, setting it apart from Ansible, which, despite widespread usage, leans more towards a code-centric approach. Infrastructure as Code tools for different project selected based on the cost,cloud platform, performance, other integration tools like Jenkins, git, github ect.

"Deploying Terraform and Ansible to Implement Hadoop Architecture"[9] paper discusses the challenges associated with managing huge amounts of data in the Big Data field and offers an automated solution utilising DevOps tools, including Terraform and Ansible. Terraform allows the team to easily and effectively provide cloud resources. Ansible is essential to automation and configuration management. Terraform is the first step in the workflow since it handles infrastructure setup on the AWS cloud. The Terraform setup starts with the 'terraform init' command, and a preview of changes is shown with the 'terraform plan' command. The infrastructure that has been planned for deployment is then deployed by the 'terraform apply' command. By creating AWS instances depending on the specified Terraform configuration, this dynamic provisioning method makes it possible to modify the infrastructure to meet the demands of the project.Ansible takes over configuration management of the Hadoop cluster deployed on the provided AWS instances after the infrastructure created. Ansible playbooks manage basic tasks like installing Java, downloading and extracting Hadoop, and configuring all of the Hadoop HDFS and MapReduce nodes Ansible shows to be a suitable option for effectively controlling a large number of servers by utilising SSH for connectivity. This approach help to dynamically modify the number of cloud-running servers, responding in real-time to the project's changing demands.Most importantly, Ansible's independent modules are essential for providing dependability and efficiency during the configuration procedure. This complete automation approach creates a foundation for a scalable and reliable project infrastructure in along with improving operational efficiency.

The paper, "Automatic control of the quality of service contract by a third party in the Cloud Computing"[10] evaluates Quality of Service (QoS), user requirements, and services. Service Level Agreements (SLAs) that serve as a guarantee of particular services within specified times and prices between suppliers and customers. They are an indication of the level of service provided. Customers expect a service to be consistently available, safe, and backed by strong security measures, high integrity and consistent performance. Services, user needs, and QoS are all considered important aspects of quick delivery; users expect their needs to be covered in certain amounts of time. The paper discusses tiered QoS management in the cloud and explains how services are delivered in the three tiers of cloud computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The process of transforming customer demands into basic service requirements is demonstrated through a case study, illustrating the difficult task of balancing resource availability with user expectations while following to timelines. The last section discusses scheduling and integration in cloud computing, focusing on the transition from physical to virtual resource services and the resulting increase in QoS metrics. The timely delivery of services is critical to the overall success of cloud computing systems, as it is closely linked to user expectations.

In conclusion, the literature review sets the stage for the empirical study, highlighting the importance of infrastructure as code and introducing key tools in the field. While the literature review offers a good foundation, the current study adds to it by comparing the provisioning time assessments of Terraform and Ansible in real-world scenarios.The research questions and methodology provide a structured approach to assess the performance and suitability of Terraform and Ansible in real-world scenarios.

5 Main Milestones Anticipated

Date	Task
January 2023	Presentation of dissertation proposal.
January 2024 - February 2024	Develop methodology and Research on methodology to implement suitable
February 2024-March 2024	Develop suitable tests for measurements.
April 2024	Develop cloud service for comparison.
May 2024	Interim report submission.
May 2024-June 2024	Working on Report and continue learning.
July 2024	Complete draft of final dissertation for supervisor.
August 2024	Conclusion
September 2024	Dissertation Submission and Dissertation Presentation and Assessment

Table 1: Future work

References

[1] Jez Humble and David Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation* in Addison-Wesley Professional, 2010.

[2] IBM, *Infrastructure as Code.* [Online]. Available: `https://www.ibm.com/topics/infrastructure-as-code`.

[3] Ansible, *Ansible Documentation.* [Online]. Available: `https://www.ansible.com/overview/how-ansible-works`.

[4] HashiCorp, *Terraform Documentation.* [Online]. Available: `https://developer.hashicorp.com/terraform/intro`.

[5] Ankita Dalvi. *Cloud Infrastructure Self Service Delivery System using Infrastructure as Code* in IEEE 2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS),2022.

[6] Michael Howard. *Terraform — Automating Infrastructure As A Service* in arXiv.org:2205.10676v1 [cs.SE] 2022

[7] Artur Cepuc, Robert Botez, Ovidiu Craciun, Iustin-Alexandru Ivanciu and Virgil Dobrota. *Implementation of a Continuous Integration and Deployment Pipeline for Containerized Applications in Amazon Web Services Using Jenkins, Ansible, and Kubernetes* in IEEE 2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet), pp. 1 - 6, 2020.

[8] Julio Sandobalín, Emilio Insfran and Silvia Abrahão, *On the Effectiveness of Tools to Support Infrastructure as Code: Model-Driven Versus Code-Centric* in IEEE Access, Vol 8, pp. 17734 - 17761, 2020.

[9] Manu Gupta, Mandepudi Nobel Chowdary, Sankeerth Bussa and Chennupati Kumar Chowdary,*Deploying Hadoop Architecture Using Ansible and Terraform* in 2021 5th International Conference on Information Systems and Computer Networks (ISCON), Mathura, India, pp. 1-6,2021.

[10] Adil Maarouf, Abderrahim Marzouk and Abdelkrim Haqiq, *control of the quality of service contract by a third party in the Cloud Computing* in 2014 Second World Conference on Complex Systems (WCCS), Agadir, Morocco, pp. 599-603, 2014.