



**UNIVERSITY OF  
PORTSMOUTH**

**Final Year Project**

# **M.Sc. Computer Network Administration and Management.**

**A Virtual Private Cloud (VPC) proposed for Pallisa General Hospital, provisioned using Terraform as an Infrastructure as code (IaC) tool in Amazon Web Services (AWS).**

**By  
Otilia Linore Kawuma**

**Module Codes: M32616-2023/24-PAYEAR  
Supervisor: Dr Mani Ghahremani**

**September 2024**

<b>Table of Contents</b>	<b>4</b>
<b>Abstract</b>	<b>4</b>
<b>Acknowledgement</b>	<b>5</b>
<b>1 Introduction</b>	<b>7</b>
1.1 Research Context	7
1.2 Project Aim	7
1.3 Project Objectives	8
<b>2 Literature Review</b>	<b>9</b>
2.2 Similar Works	9
2.2.1 Infrastructure Provisioning in AWS: Virtual Private Clouds	9
2.2.2 Infrastructure as Code (IaC)	9
2.2.3 Current state of Terraform	10
2.2.4 How Terraform works.	11
2.2.4.1 Image 1, showing how terraform works	11
2.2.4.2 Image 2, showing a standard terraform workflow	12
2.4 Summary	12
<b>3 Methodology</b>	<b>13</b>
3.1 Agile Methodology	13
3.1.1.1 Core Principles of Agile.	13
3.1.1.2 Agile sub-categories	13
3.1.1.3 Suitability of Agile Methodologies, with a focus on scrum and kanban	14
Kanban versus Scrum	14
3.1.1.4 Kanban vs. Other Agile Methods Mentioned above	14
3.1.2 Software Development Model Used	15
3.2 Project Planning	16
3.2.1 Initial Plan	16
3.2.2 Actual Plan	16
3.2.3 Reasons for Plan Change	16
3.3 Ethical Considerations	17
3.3.1 Ensuring Compliance and Mitigation Strategies.	17
<b>4 Requirements and Analysis</b>	<b>18</b>
4.1 Requirement Gathering	18
4.1.1 Image 1: Showing low performance of Pallisa district health sector, including Pallisa Hospital, extracted from (Local Government Performance Assessment, n.d).	18
4.1.2 Existing Infrastructure System	19
4.1.2 Proposed VPC Requirements	19
4.1.4 Technical Feasibility	19
4.2 System Requirements	19
4.3 Requirements Analysis	19
4.3.1 Security and Compliance	20
4.3.2 Scalability and Performance	20
4.3.3 Connectivity and Integration	20
4.4 Table of Requirements	21

<b>5 Design</b>	<b>22</b>
5.1 System Architecture Overview	22
5.1.1 Initial Design	22
5.3 VPC Architecture Overview	23
5.3.1 Diagram 1 Architectural design of Pallisa Hospital's 3-tier VPC.	23
5.2 Sequence Representation of Events	24
5.2.1 Conceptual Workflow	24
5.2.2 Web flow and Network access design	25
Components involved:	25
<b>6 Implementation</b>	<b>26</b>
6.1 Iteration 1	26
6.1.1 Planning	26
6.1.2 Implementation	27
6.2 Iteration 2	27
6.2.1 Planning	27
6.2.2 Terraform Installation CLI	27
6.2.3 Install the editor and its plugins	27
6.2.4 Download and install the AWS_CLI	28
6.2.5 Configure AWS CLI credentials	28
6.3 Iteration 3	29
6.3.1 Planning	29
6.3.2 Implementation	29
6.4 Iteration 4	30
6.4.1 Planning	30
6.4.2 Implementation	30
6.5 Iteration 5	31
6.5.1 Planning	31
6.5.2 Testing the Terraform code	32
6.5.2.1 Some errors encountered in code execution	34
6.5 Review	35
6.5.1 Below is the VPC created in Amazon Web Services using terraform code.	35
6.5.1 Terraform destroy	36
<b>7 Evaluation</b>	<b>37</b>
7.1 Table of Requirements	37
4.4 Table of Requirements	37
7.2 Limitations in using IaC tools like Terraform	38
7.2.1 Key Limitations of Building Infrastructure in AWS	38
1. VPC and Subnet Limits:	38
7.2.2 Mitigating Limitations	39
<b>8 Conclusion</b>	<b>40</b>
8.2 Future Work	40
<b>References</b>	<b>41</b>
Appendix I — Certificate of Ethics Review	43
Appendix II — Sample code of Pallisa Hospital's VPC.	44

# Table of Contents

## Abstract

The use of automation tools to manage applications in the cloud has been on the rise, Terraform an Infrastructure as code tool is widely used for provisioning infrastructure in multiple cloud providers. For Pallisa General Hospital, Terraform will be leveraged to build a robust, secure and scalable Virtual Private Cloud (VPC) in Amazon Web Services (AWS).

The goal is to address the hospital's current IT infrastructure challenges, including poor data and information security, inefficient resource management, and scalability issues. By leveraging cloud computing and Infrastructure-as-Code (IaC) tools, the project seeks to enhance the Hospital's IT resilience, security, and scalability while reducing operational costs. The objectives include architecting a custom VPC, using HashiCorp Configuration Language (HCL), enhancing security and compliance with Ugandan healthcare regulations, and improving resource scalability.

The project will be structured in the sense that a literature review is followed, requirements gathering, architecture design, Terraform code implementation, and evaluation. The outcome is expected to serve as a benchmark for rural hospitals, demonstrating how cloud computing and IaC tools can transform healthcare infrastructure and improve patient care.

<b>Word count: 12000</b>
--------------------------

# Acknowledgement

I thank my supervisor, Dr Mani Ghahremani, for his invaluable guidance, support, and insightful feedback throughout this dissertation. Your expertise and encouragement have shaped my research and understanding of the subject.

I am thankful to God above all else, I am persuaded that he led me to and through this opportunity and brought my way all the help I needed to finish this project well.

I sincerely appreciate my family and friends, especially my husband Moses Kawuma, for his unconditional love and support.

This page intentionally left blank

# 1 Introduction

## 1.1 Research Context

By 2025, 51% of IT spending will move to the cloud (Jayaraman, 2024). As such, it is no surprise that many companies and organisations big or small are starting to embrace cloud computing. Pallisa General Hospital is no exception.

Pallisa Hospital's legacy IT infrastructure could be more efficient, currently, it poses data security risks and needs to improve its ability to manage growing patient data and service demands. Manual IT resource management is labour-intensive, error-prone, and incurs high operational costs.

To address these challenges, implementing a Virtual Private Cloud (VPC) on Amazon Web Services (AWS) using Terraform as an infrastructure-as-code (IaC) tool for efficient, consistent provisioning and automated deployment is proposed.

This project proposal will demonstrate how cloud computing and IaC can transform healthcare infrastructure by enhancing Pallisa Hospital's IT resilience, security, and scalability. It will set a benchmark for other rural hospitals and improve patient care and operational efficiency.

Cloud-based solutions' scalability, interoperability, and security solve immediate problems and set the healthcare sector up for future advancements in health and care provision, information management and collaboration amongst professionals across various departments (Tak, 2023).

## 1.2 Project Aim

This project aims to design, implement, and provision a Virtual Private Cloud (VPC) for Pallisa Hospital using Terraform on Amazon Web Services (AWS).

The goal is to establish a robust, secure, and scalable cloud infrastructure that enhances data management, optimizes resource utilisation and strengthens security. The project will demonstrate how Terraform can be used to automate and streamline the provisioning and management of cloud resources, thus ensuring consistency, facilitating easier system updates, reducing manual errors and easily scaling infrastructure.

In summary, the dissertation will;

- Architect a VPC tailored to Pallisa Hospital's needs.
- Provision the VPC using Terraform on AWS.
- Enhance security and compliance with healthcare regulations in Uganda.
- Improve resource scalability and reduce operational costs in handling data.

## 1.3 Project Objectives

Initially, a literature review will be conducted to survey existing research, developments, and tools in the realm of cloud infrastructure and the use of Terraform as an infrastructure-as-a-code tool. This foundational research will inform the subsequent planning phase, where decisions on the project's execution strategy will be solidified.

Following the planning stage, the artefact's requirements will be determined, outlining essential features that will facilitate scalability, security, and data management capabilities to align with the hospital's operational demands. This will be done with AWS's well-architected Infrastructure framework mindset, which outlines six pillars, including security, reliability, performance efficiency, cost optimization, and sustainability.

The next phase involves designing the architecture of the VPC, which will be visualized through an architectural diagram to provide a clear blueprint for construction.

Various resource interactions will be observed in the build phase using the AWS console. Terraform's existing modules and building blocks will be leveraged, and the HashiCorp language (HCL) will be used to write Terraform code to produce a custom VPC module to build, automate and manage cloud resources. Terraform keeps a state file of current resources, this ensures consistency across deployments and facilitates updates while minimizing manual errors.

To evaluate the artefact's performance and efficacy, Terraform's workflow is to be used as a guideline. This evaluation will inform how well the provisioned VPC aligns with AWS's well-architected framework of operational excellence, security, reliability, performance efficiency, cost optimization and the predefined requirements and expectations of the hospital.

The implementation stage will be realized within AWS's command line interface, where the designed infrastructure will be brought to life. Finally, an extensive evaluation of the artefact will be done to ensure it meets all performance benchmarks.

In summary, this project's goal is not only to deliver a robust VPC for Pallisa Hospital, but also to set a precedent for future cloud infrastructure projects in similar organisations. It envisions offering recommendations for enhancements and exploring potential areas for future work to keep pace with technological advancements and evolving healthcare needs.



## 2 Literature Review

Infrastructure as Code (IaC) tools like Terraform, Ansible, and Lambda functions entail the provisioning of cloud resources through machine-readable files, rather than relying on manual configuration tools. In today's fast-paced world, IaC is crucial for organisations of all sizes, allowing developers to create resources efficiently and save time (Kavas, 2023).

A Virtual Private Cloud is an example of such a resource. In the AWS cloud, a VPC is an Infrastructure as a service offering (IaaS) that allows individuals and organisations to build virtual private networks. Resources can be provisioned in this logically isolated space, including Amazon Relational Database Services (RDS) instances, Amazon Elastic Cloud Compute (EC2) instances, public and private Subnets, Route tables, Internet gateways, Network Address Translation (NAT) gateways, Elastic network interfaces (ENI) and many more. This network can also be configured to connect to other VPCs and on-premise resources securely via direct connection (Anthony, 2017).

As an open-source Infrastructure as Code (IaC) tool developed by HashiCorp, Terraform automates the provisioning, configuration, and management of infrastructure resources across various cloud environments spanning several availability zones (AZs). It is possible to work across multiple providers such as AWS, Azure, and Google Cloud, on-premises data centres, and other service providers. To define the state of desired resources, Terraform defines its configurations using a declarative language, Hashicorp (Leveraging Terraform as a Battle-Tested Solution, 2024).

### 2.2 Similar Works

#### 2.2.1 Infrastructure Provisioning in AWS: Virtual Private Clouds

In their work to develop a platform as a service (PaaS) environment in AWS for medical imaging analytics in 2018, Ivanova, P. Borovska and S. Zahov used Terraform as their Infrastructure as code (IaC) tool of choice. They created a production-grade platform in the Amazon Cloud. The environment supported all necessary software tools and packages whilst remaining secure, fast and powerful yet also easy to manage and economical to maintain. The PaaS infrastructure was built in “3.25” minutes. Using the Amazon console could have taken days or even weeks to complete. (Ivanova, 2018)

#### 2.2.2 Infrastructure as Code (IaC)

In Africa, especially in Kenya, Infrastructure-as-Code (IaC) presents overwhelming advantages for cloud-based business continuity, especially for Institutions of Higher Learning (IHLs). Optimizing IaC tools such as Terraform permits institutions to minimize expenses related to physical backup facilities, streamline workflows, maintain consistency, and accelerate cloud migration deployments. However, despite these advantages, Kenyan IHLs have been hesitant to embrace cloud-based business continuity solutions due to various obstacles like limited bandwidth and high access costs.

Muthoni S, Okeyo G, and Chemwa G highlighted that Infrastructure as code effectively addresses critical challenges in current deployments in their research works encompassing ninety-two university personnel with cloud computing expertise. This comprises elevated human resource costs, complex processes, heightened risk of human error, and substantial maintenance expenses. Their experimental findings utilized Ansible and Terraform to demonstrate the efficacy of IaC in simulating recovery from an on-premises data centre to Amazon Web Services (AWS).

The study hypothesized a framework grounded in People, Process, and Technology (PPT) principles. It aimed to assist Institutions of Higher Learning (IHLs) to efficiently provision cloud-based business continuity solutions by leveraging the benefits of IaC tools like Terraform. This approach seeks to overcome adoption barriers and enhance operational resilience in the higher education sector and other Institutions (Muthoni et al., 2021).

Odeyinka J considered Terraform as a defacto IaC programming language to provision a two and three-tier architecture in AWS. Leveraging on AWS's automation services like AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and AWS CodePipeline, he wrote code that seamlessly integrates source code management, automated build processes, comprehensive testing frameworks, and reliable deployment mechanisms using terraform modules. This allowed for the creation and usage of abstraction on top of the resources set. His architectural marvel harnessed the power of AWS services and tools to create a robust and highly customizable pipeline, tailored to meet the unique needs of any organization (Odeyinka, 2024).

### 2.2.3 Current state of Terraform

Terraform, introduced in 2017, transformed IT infrastructure management by simplifying deployments across multiple cloud providers with code. Despite initial complexities reported by some organisations, especially those using cloud computing, containerization, and other cutting-edge infrastructure technologies.

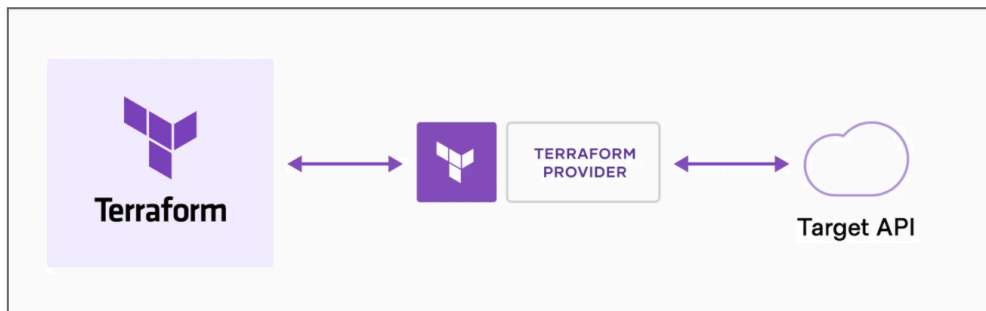
Terraform definitively reduces time, effort, human interaction, and errors, which were initially a major obstacle. It has been and continues to be instrumental in improving efficiency in managing numerous projects simultaneously across multiple cloud platforms.

Mehdi and Walia encourage IT teams including Infrastructure engineers, DevOps teams and System Admins seeking a frugal and automated method of provisioning and managing infrastructure to always consider Terraform. The application deployment process using Terraform allows for efficient provisioning and management of cloud resources, be it in AWS, Microsoft Azure or Google Cloud. With the ability to define infrastructure as code, the deployment becomes standardized and easily repeatable, ensuring consistency across different environments.

Terraform guarantees a robust infrastructure, and resources built using Terraform are easy to secure. With increasing focus on security and compliance today, Terraform enables the implementation of secure network architectures, proper access controls like IAM roles, and well-defined security groups to protect against potential threats (Mehdi & Walia, 2024).

## 2.2.4 How Terraform works.

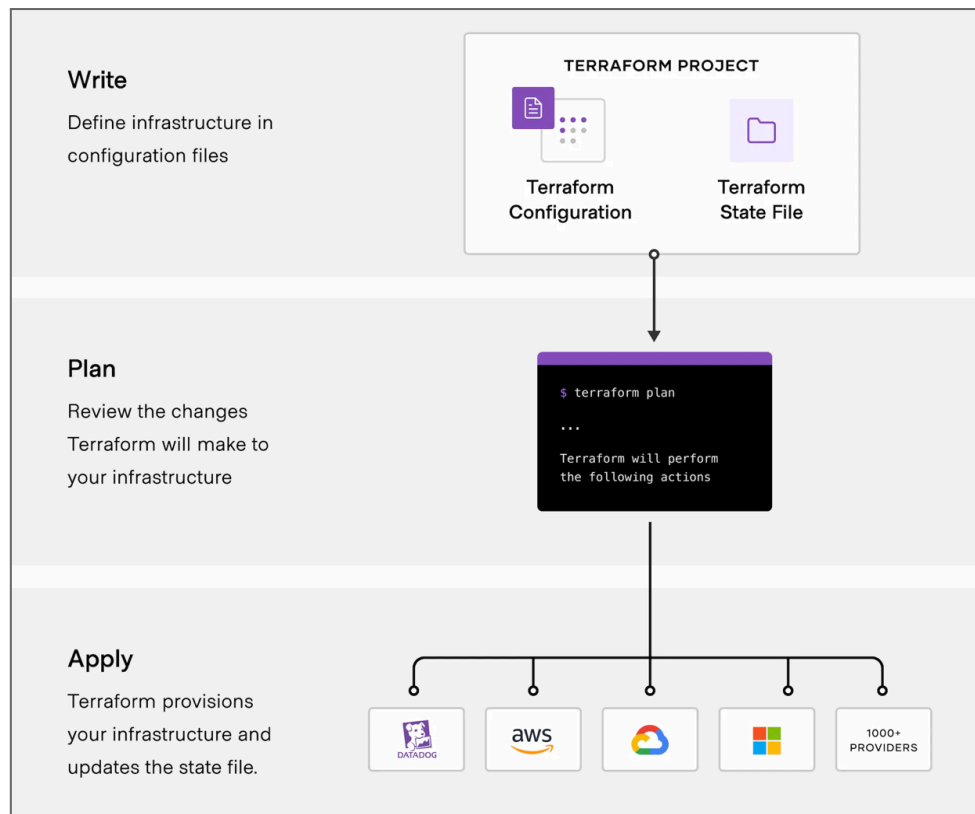
To manage and create resources on AWS, Terraform uses application programming interfaces (APIs). To work with any virtual platform with accessible APIs, Terraform makes use of providers as illustrated in [images 1](#) and [2](#) below. HashiCorp and the Terraform community have already written thousands of providers to manage many resources and services. These are publicly available in the [Terraform registry](#) including Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, Datadog, and many more (Hashicorp Inc., 2018).



### 2.2.4.1 Image 1, showing how terraform works

The core Terraform workflow consists of three stages, as shown in [image 2](#):

- **Write:** You define resources, which may be across multiple cloud providers and services. For example, you might create a configuration to deploy an application on virtual machines in a Virtual Private Cloud (VPC) network with security groups and a load balancer.
- **Plan:** Terraform creates an execution plan describing the infrastructure it will create, update, or destroy based on the existing infrastructure and your configuration.
- **Apply:** On approval, Terraform performs the proposed operations in the correct order, respecting any resource dependencies. For example, if you update the properties of a VPC and change the number of virtual machines in that VPC, Terraform will recreate the VPC before scaling the virtual machines (Hashicorp Inc, 2018, para)



2.2.4.2 Image 2, showing a standard terraform workflow

## 2.4 Summary

Pallisa Hospital stands to benefit significantly from using Terraform to provision a production-grade Virtual Private Cloud (VPC) in AWS. By leveraging Infrastructure-as-Code (IaC), the hospital can automate the setup of a secure, scalable, and cost-effective cloud infrastructure. Infrastructure-as-code tools like Terraform are built-in integrated development environments (IDEs) like VS Code. These further allow for collaboration and knowledge sharing among diverse teams.

Terraform also allows for rapid deployment, reducing the time and manual effort typically needed, thus minimizing operational inefficiencies and human errors. Additionally, it supports the implementation of robust security measures, such as secure network architectures and access controls. These advantages make Terraform an ideal choice for improving IT resilience and business continuity in healthcare settings, even in Pallisa Hospital.

## 3 Methodology

### 3.1 Agile Methodology

Agile methodologies are relevant to the topic of Virtual Private Cloud Provisioning with Terraform in AWS, as they provide frameworks for efficient and iterative development of infrastructure.

Agile, an iterative approach to software development and project management, has principles, rooted in customer satisfaction, stakeholder engagement, and collaborative growth. Agile emphasises flexibility, adaptability and quick delivery of solutions. The delivery of software takes precedence over exhaustive documentation, while also collaboration with clients to understand requirements supersedes contract negotiation (Tetteh, 2024).

#### 3.1.1.1 Core Principles of Agile.

1. **Collaborating with Clients;** Engaging stakeholders and customers throughout the project to ensure the delivered solution meets their needs.
2. **Embracing changes;** In responding to changes and requirements, even late in development, to provide a competitive advantage.
3. **Iterative Development:** Delivering working systems frequently, with a preference for shorter timescales.
4. **Continuous Improvement:** Regularly reflecting on how to become more effective and adjusting behaviours accordingly.

#### 3.1.1.2 Agile sub-categories

Agile methodologies include:

1. Kanban
2. Scrum
3. Adaptive Software Development (ASD)
4. Agile Unified Process (AUP)
5. Crystal Methods
6. Dynamic Systems Development Methodology (DSDM)
7. eXtreme Programming (XP)
8. Feature Driven Development (FDD)
9. Lean Software Development

### 3.1.1.3 Suitability of Agile Methodologies, with a focus on scrum and kanban

#### **Kanban versus Scrum**

Scrum, which is a collaboration framework in Agile organizes work into fixed-length iterations referred to as sprints, these last two to four weeks. Each Sprint aims to deliver a potentially shippable product increment in no longer than a month.

Scrum makes use of the specific roles listed below;

1. The Scrum Master
2. Product Owner
3. Ceremonies include Sprint Planning, Daily Stand-ups, Sprint Reviews, and Retrospectives.

These components provide a structured, disciplined approach to managing complex projects, ensuring all team members are aligned and focused on delivering incremental progress.

On the other hand, Kanban is a more flexible methodology of Agile that emphasizes visualizing work and workflows for continuous delivery and process improvement.

It leverages a visual board to manage workflow, allowing team members to see the status of tasks in real time. Kanban uses a pull model that pulls work items from a product backlog into a steady flow, this famous practice limits Work in Progress (WIP), thus optimizing efficiency and preventing bottlenecks. Unlike Scrum, Kanban does not prescribe specific roles or iterations, allowing teams to adapt quickly to changing requirements and priorities.

#### **Suitability for the 3-Tier VPC Project**

For our 3-tier VPC project, Kanban is preferable for the reasons below:

1. The Kanban board gives a clear visual representation of tasks, making it easier to track progress and identify bottlenecks. This transparency is essential for managing complex infrastructure projects.
2. Adaptability and Flexibility: Kanban's visual boards accord teams an adaptability edge to system changes without waiting for the end of iterations. This flexibility is palatable in cloud infrastructure projects where requirements can change rapidly.
3. Continuous Improvement: Kanban's focus on continuous improvement aligns well with the iterative nature of infrastructure development, allowing the team to refine processes and improve efficiency over time.
4. No Prescribed Roles: Kanban's lack of specific roles reduces the need for additional training or restructuring, allowing the team to operate within their existing structure.

### 3.1.1.4 Kanban vs. Other Agile Methods Mentioned Above

Adaptive Software Development (ASD) emphasizes adaptive cycles and collaboration. While it offers flexibility, Kanban's visual approach and focus on flow make it more effective for managing infrastructure tasks.

Extreme Programming (XP) hinges on close collaboration with customers, frequent releases, and continuous feedback. While XP is beneficial for software development, its focus on rapid iterations and customer involvement may not be as applicable to infrastructure projects like VPC provisioning.

Feature-Driven Development (FDD) is a model-driven process that focuses on delivering features. It involves detailed planning and design, which may not provide the flexibility required for dynamic infrastructure projects.

Lean Software Development focuses on eliminating waste and optimizing processes. While Lean shares some principles with Kanban, such as continuous improvement, Kanban's visual management and work-in-progress (WIP) limits make it more suitable for managing complex workflows in infrastructure projects.

Agile Unified Process (AUP) is a simplified version of the Rational Unified Process (RUP) that incorporates Agile principles. AUP's structured approach may not provide the same level of adaptability as Kanban for infrastructure projects.

Dynamic Systems Development Method (DSDM) is a comprehensive Agile framework that emphasizes project governance and control. While DSDM is effective for large projects, its complexity may not be necessary for the VPC project.

### 3.1.2 Software Development Model Used

For this project, the Kanban approach is preferred due to Kanban's strengths in flexibility, visual management, continuous improvement, and minimal disruption, making it the most suitable Agile methodology for the 3-tier VPC project in Pallisa Hospital.

Its ability to adapt to changing requirements and maintain a steady flow of work aligns well with the dynamic nature of cloud infrastructure provisioning.

By deciding on Kanban, the project's complexities can effectively be managed while ensuring timely delivery and alignment with the hospital's goals, as supported by the findings in the journal by Tetteh, S. G. in [Agile Methodology](#) above.

## 3.2 Project Planning

A Gantt chart's visual timeline, allowed for task dependencies to be identified from inception to finish.

### 3.2.1 Initial Plan

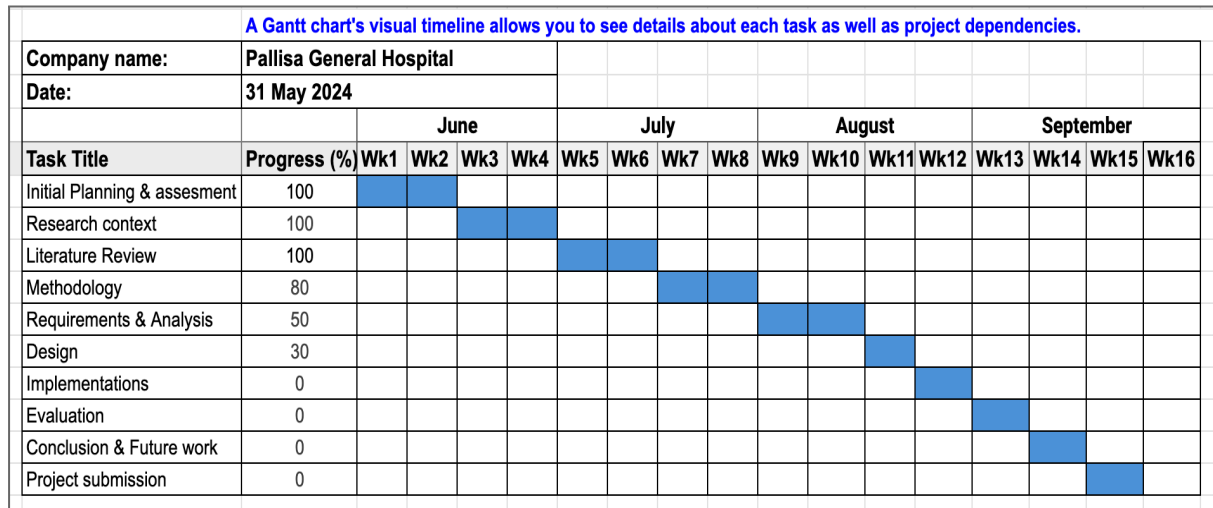


Figure 1: Gantt chart showing the initial project plan and task dependencies

### 3.2.2 Actual Plan

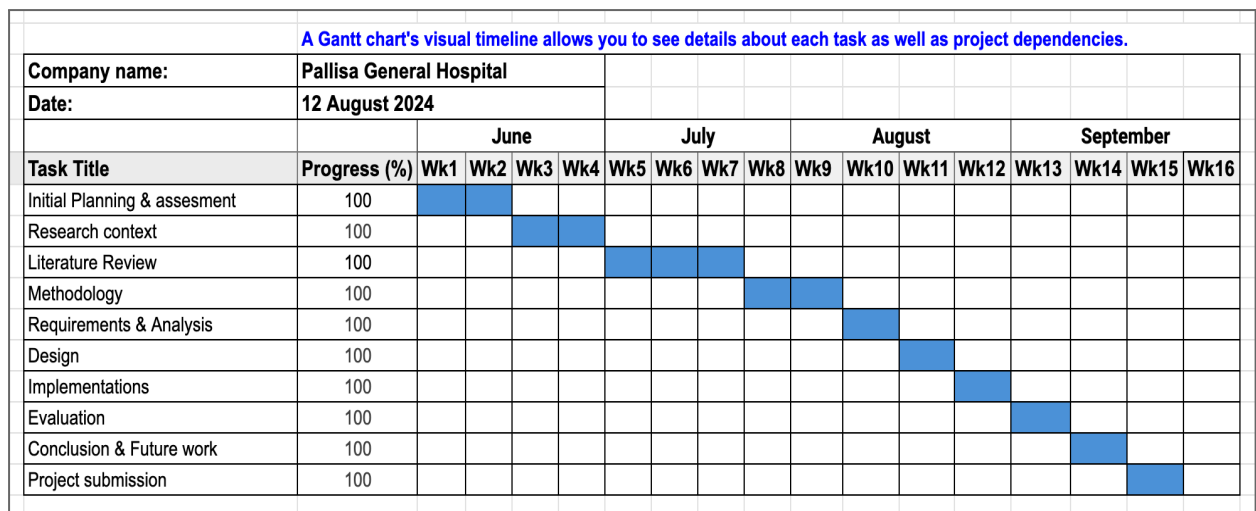


Figure 2: A Gantt chart showing the final project plan, task dependencies and changes.

### 3.2.3 Reasons for Plan Change

- All planning was done according to the initial plan with minor changes, apart from the literature review, which took a week more due to the diverse sources and the need to cover pertinent literature to ensure depth, breadth, and credibility.
- Requirements and Analysis took only one week instead of two due to the good understanding of the subject gained in the literature review and methodology, in the end, the project was done in the initially allocated time of 15 weeks.



### 3.3 Ethical Considerations

The design of the VPC with Terraform is a protective mechanism for safeguarding the state of infrastructure and data storage on servers and virtual machines.

Terraform keeps a state file of the desired and actual cloud infrastructure, in this setup, the current infrastructure state will securely be stored in an s3 bucket and locked with a DynamoDB. This enhances security by providing encryption, versioning, and access control, preventing unauthorized access, impromptu changes and infrastructure drift.

To safeguard patient information, data encryption and robust access controls are crucial. Obtaining informed consent from the Uganda Communication Commission (UCC) for data collection and processing will be key.

Unauthorized access to leaked data could lead to identity theft or fraud. This project minimizes this risk through encryption, stringent access controls using IAM roles, regular audits, and compliance with healthcare regulations.

The VPC design aligns with Ministry of Health standards to securely manage medical records. Considering Uganda's socio-economic vulnerabilities, the VPC infrastructure supports equitable healthcare access and does not worsen existing inequalities.

#### 3.3.1 Ensuring Compliance and Mitigation Strategies.

In summary, to guarantee compliance with the ethical considerations:

- Apply Security measures like Resource Based Access Control (RBAC), and encryption of data at rest and in transit.
- Use of firewalls security groups at resource levels and network access control lists at subnet levels.
- Educate employees through training on data protection laws and best approaches to ensure compliance and ethical handling of data.

#### **Facilities and Resources:**

- Personal laptops can be allowed under a structured bring-your-own-device policy (BOYD).
- Access to a remote computer is possible as a backup route to access Hospital services through securely configured VPNs
- Use of AWS console accounts attached to predefined IAM roles and policies within the Hospital Network.

## 4 Requirements and Analysis

In designing the 3-tier VPC, Terraform is the Infrastructure as Code (IaC) tool chosen. A good understanding of the hospital's needs and technical requirements was considered and analysed, with a focus on efficiency and infrastructure agility, as illustrated below;


### 4.1 Requirement Gathering

Here, the Hospital's stakeholders were identified, who are mainly the government of Uganda and Pallisa district local government.

The current infrastructure is majorly manual, inhibiting the scalability of systems, efficiency and secure handling of patients, leading to poor performance of the healthcare sector in Pallisa as shown in [Image 1](#) below.

The local government performance assessment report for Pallisa district, which showed low performance in the healthcare sector in the last three years, was looked at to see if the proposed VPC would be suitable.

This further informed the objectives of Infrastructure capacity and design strategies in the hospital and other health and care providers, as shown in the image below.

	
<b>Local Government Performance Assessment</b>	
Pallisa District	
(Vote Code: 548)	
Assessment	Scores
Accountability Requirements	67%
Crosscutting Performance Measures	66%
Educational Performance Measures	92%
Health Performance Measures	64%
Water Performance Measures	59%

*4.1.1 Image 1: Showing low performance of Pallisa district health sector, including Pallisa Hospital, extracted from (Local Government Performance Assessment, n.d).*

### 4.1.2 Existing Infrastructure System

From the information gathered, the existing infrastructure is having the problems below:

- Lack of remote access to Hospital computers.
- Uncontrolled and chaotic growth of data.
- Wastage of resources.
- Vulnerabilities in data access and Information management.

### 4.1.2 Proposed VPC Requirements

The main objective of proposing a virtual private cloud is to write the Hospital's Infrastructure in code form that will manage hospital services to overcome the above shortfalls:

- Establish a robust, secure, and scalable cloud infrastructure.
- Improve data management.
- Optimize resource utilisation.
- Strengthen security.

### 4.1.4 Technical Feasibility

This aimed at evaluating the main features of the system proposed, the requirements of the output and the long-term effects of the coded infrastructure, below are some questions asked:

- Will the new infrastructure easily scale in and out?
- What systems, services, and applications will the VPC run?
- How easy is it to secure data and information in the VPC subnet layers?
- How will the current infrastructure work with the new until full adoption?
- Are the IT staff technically ready to manage and administer the new system?

It was then decided that Terraform, a popular IaC tool, would be used as it addresses most of the questions above. Generally, the proposed VPC is feasible for development.

## 4.2 System Requirements

The virtual private cloud is envisioned to be accessed remotely via SSH or the Amazon console using pre-configured user accounts and IAM roles. The hospital staff will be able to access the virtual machines at any time and from anywhere.

## 4.3 Requirements Analysis

The 3-tier VPC architecture will consist of the following layers:

- Web Layer: This includes web servers that handle user interface and client interactions.
- Application Layer: This layer processes business logic and handles data processing tasks.
- Data Layer: This involves databases and data storage systems that manage data persistence mechanisms like Elastic Block Storage (EBS), Amazon s3, Elastic File Storage (EFS) and Amazon Relational Databases (RDS).

Depending on the services hosted, the server machines would predominantly run on the Linux operating system.

#### 4.3.1 Security and Compliance

- To ensure the confidentiality, integrity, and availability of sensitive patient data, data in the database tier will be protected by security groups, key locks and IAM roles. As discussed in the [ethical consideration section](#) and [Requirements Analysis](#).
- Strict access control measures ensure that only authorized personnel and systems can access the cloud environment and databases. This includes using Multi-Factor Authentication (MFA) for access to the AWS console, security groups, and Network Access Control Lists (NACLs) to manage traffic flow.
- To ensure the secure handling of patient data, the VPC design complies with relevant healthcare regulations and standards, such as the Uganda Data Protection and Privacy Act, 2019. This Act regulates the collection, processing, and storage of personal data to protect the privacy of individuals.

#### 4.3.2 Scalability and Performance

- The VPC is designed to accommodate future growth, allowing the hospital to scale its IT resources up or down based on demand. Sufficient IP address spaces and scalable services like AWS Auto Scaling are planned for.
- Performance: Choosing appropriate availability zones will enable the VPC to deliver high performance for the hospital's critical applications, ensuring low latency and high availability.

#### 4.3.3 Connectivity and Integration

- Hybrid Connectivity: The VPC will support hybrid connectivity for existing on-premises systems to enable seamless integration between on-premises and cloud resources. This can be achieved using AWS Direct Connect or VPN connections.
- VPC peering facilitates secure and efficient connectivity with other VPCs in the Ministry of Health and other local government departments.

## 4.4 Table of Requirements

Requirements table to guide VPC development, aiding in prioritization and tracking.

ID	Description	Priority	Justification
1	Open an AWS account	Must have ▾	Necessary to perform any API calls in the AWS cloud
1.1	Knowledge of AWS services and how Terraform works	Must have ▾	Faster coding and a good understanding of terraform
2	Download and install the Terraform command line interface (CLI)	Must have ▾	Quick access to Terraform resources
2.1	Download and install the AWS command line interface	Must have ▾	Quick access to CLI and credential files
2.3	Install plugins for HashiCorp Terraform, AWS CLI configure,	Must have ▾	Syntax auto-completion
2.4	Configure AWS credentials	Must have ▾	To authorize terraform access rights and credentials
3	Build VPC manually in the AWS console	Could have ▾	Understand console interaction of resources in the AWS cloud
4	Configure the s3 bucket for state file locking	Must have ▾	For terraform state storage and locking
5	Write the Terraform VPC modules	Could have ▾	For re-usability
5.1	Initialize the Terraform project folder	Must have ▾	To download providers
5.2	Review and validate the infrastructure	Should have ▾	To check and verify the code
5.3	Deploy the VPC infrastructure	Must have ▾	Provisions required state
5.4	Confirm/ test infrastructure	Should have ▾	Verifying created infrastructure
5.5	Access any applications deployed in the VPC environment	Could have ▾	Further testing and verification
6	Push project details to GitHub	Should have ▾	Safekeeping, reference, and future use

# 5 Design

## 5.1 System Architecture Overview

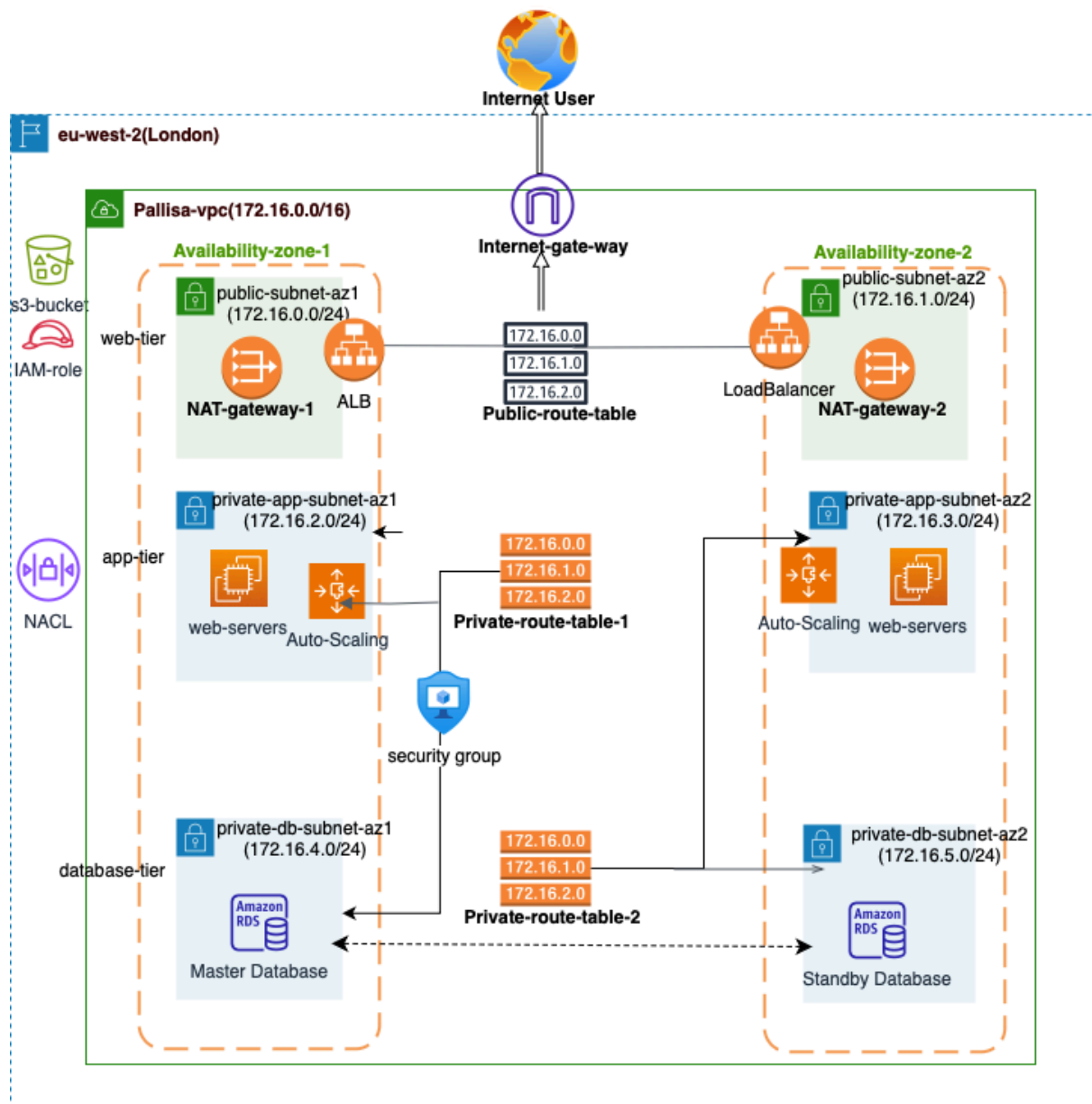
The VPC design includes an internet gateway, subnets, route tables, security groups, Network Address Control lists (NACL) and NAT gateways as illustrated in [diagram 1](#) below. These components ensure high performance, low latency, and high availability for critical hospital applications, with secure connectivity to the internet and on-premises data centres.

### 5.1.1 Initial Design

1. Presentation (Web) Tier: This tier is made up of the public subnets and includes resources like load balancers, and bastion hosts that require internet access. It also covers the internet gateway that connects the VPC to the internet, while route tables direct traffic efficiently, security groups and network access control lists regulate traffic at each tier. A NAT gateway can also be found here, it allows secure internet access for the instances in private subnets. The architecture also supports Virtual Private Connectivity (VPN) to the hospital's on-premises data centre, ensuring a safe and seamless network extension into the cloud. These components manage incoming traffic and provide secure access to the application tier.
2. Application (App) Tier: Located in private subnets, it houses the hospital's critical applications that process business logic and are isolated from direct internet access to enhance security.
3. Database (db) Tier: Contains databases and storage systems mentioned in [Requirements Analysis](#) for data protection and integrity, and does not have direct internet access

## 5.3 VPC Architecture Overview

Pallisa Hospital's three-tier reference architecture



5.3.1 Diagram 1 Architectural design of Pallisa Hospital's 3-tier VPC.

## 5.2 Sequence Representation of Events

### 5.2.1 Conceptual Workflow

The conceptual workflow below shows the basic stages for users accessing the VPC.

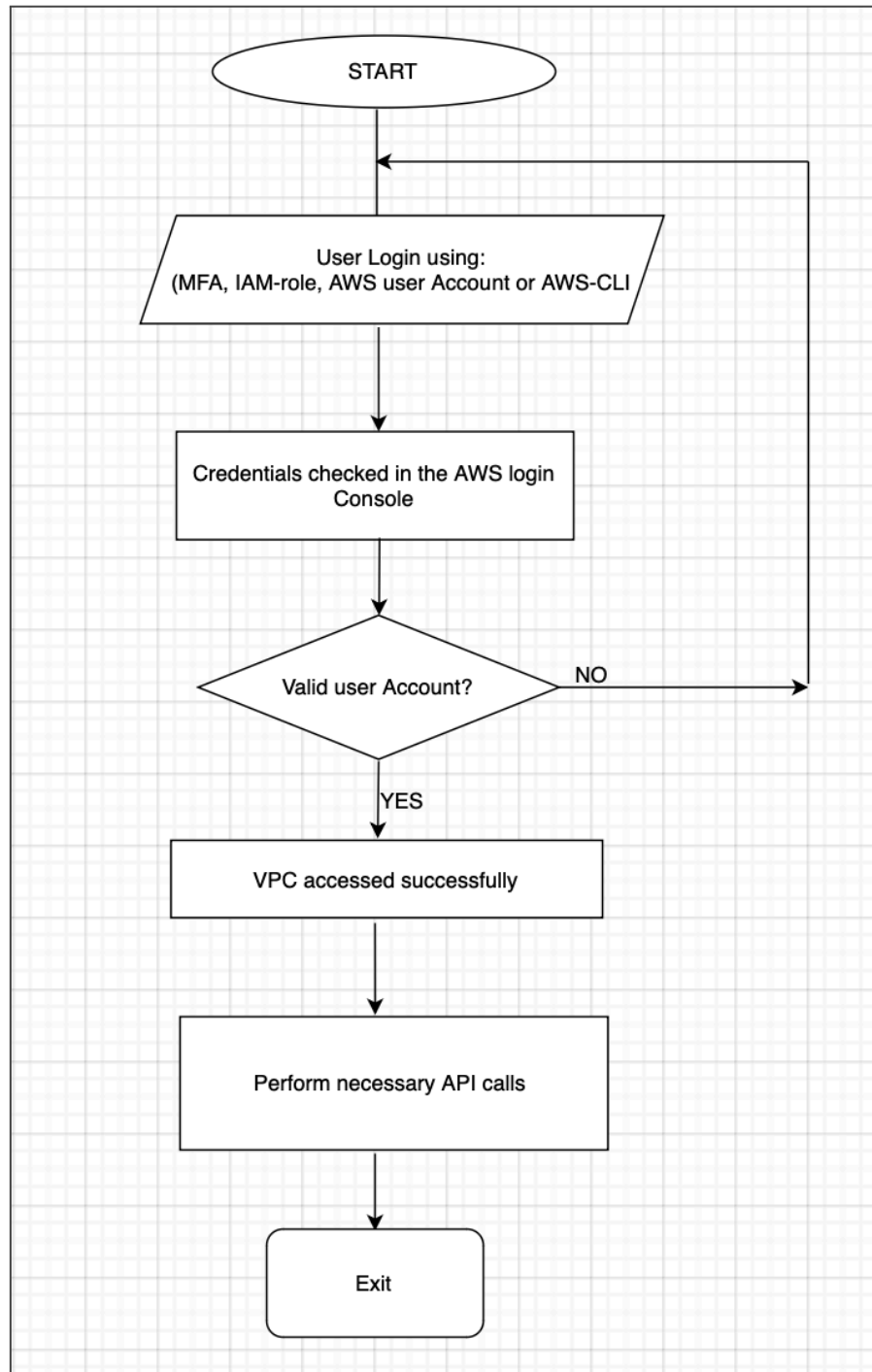
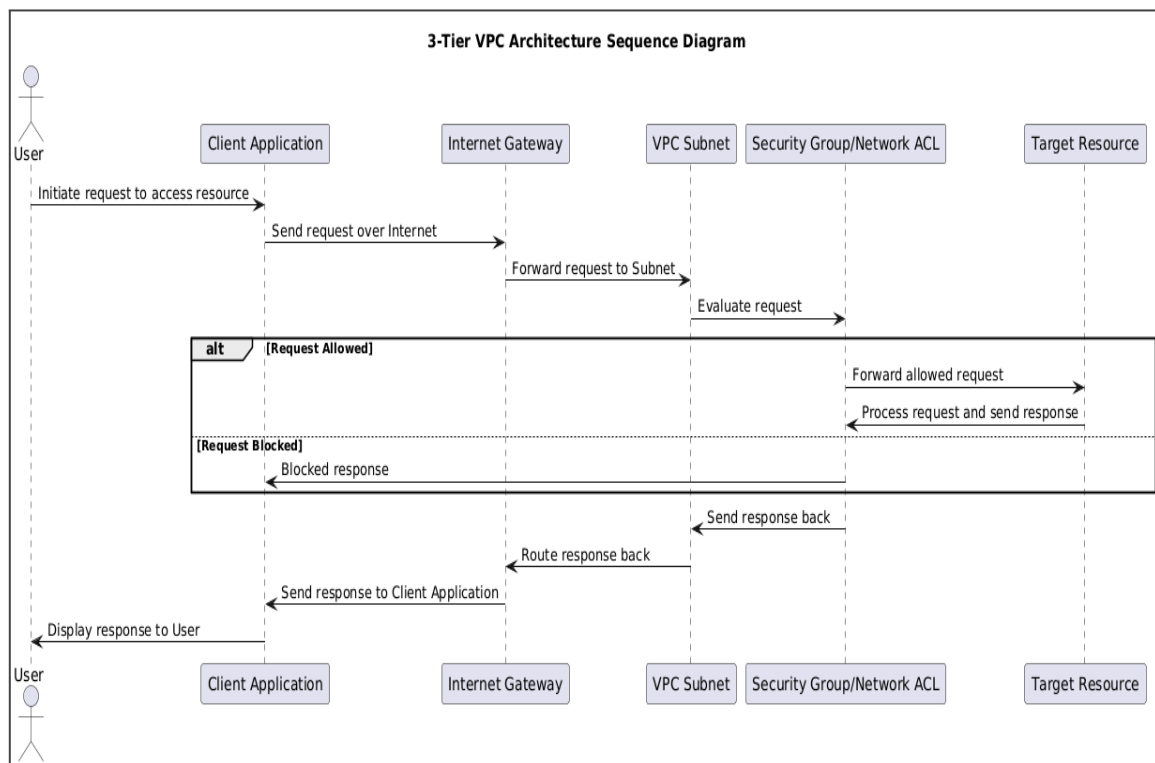


Diagram 2 Conceptual workflow, drawn using edraw.io



## 5.2.2 Web flow and Network access design



*Diagram 3 web and network flow, generated using a PlantUML tool*

Components involved:

1. User
2. Client Application (e.g., a web browser or a software tool)
3. Internet Gateway
4. Virtual Private Cloud (VPC)
5. Subnets (Public/Private)
6. Security Group/Network ACL
7. Target Resource (e.g., EC2 instance, database)

### Sequence of Actions:

1. A user initiates a request to access a resource within the VPC via the Client Application.
2. The Client Application sends the request over the Internet.
3. The request reaches the Internet Gateway, which is the entry point to the VPC from the Internet.
4. The Internet Gateway forwards the request to the appropriate Subnet within the VPC, based on routing rules.
5. The request is then evaluated by the Security Group/Network ACL associated with the Subnet to determine if it's allowed or blocked.
6. If the request is allowed, it reaches the Target Resource (e.g., an EC2 instance or a database) within the VPC.

7. The Target Resource processes the request and sends a response back through the Security Group/Network ACL.
8. The response is routed back to the Client Application via the Internet Gateway.
9. Finally, the **User** receives the response on their **Client Application**.

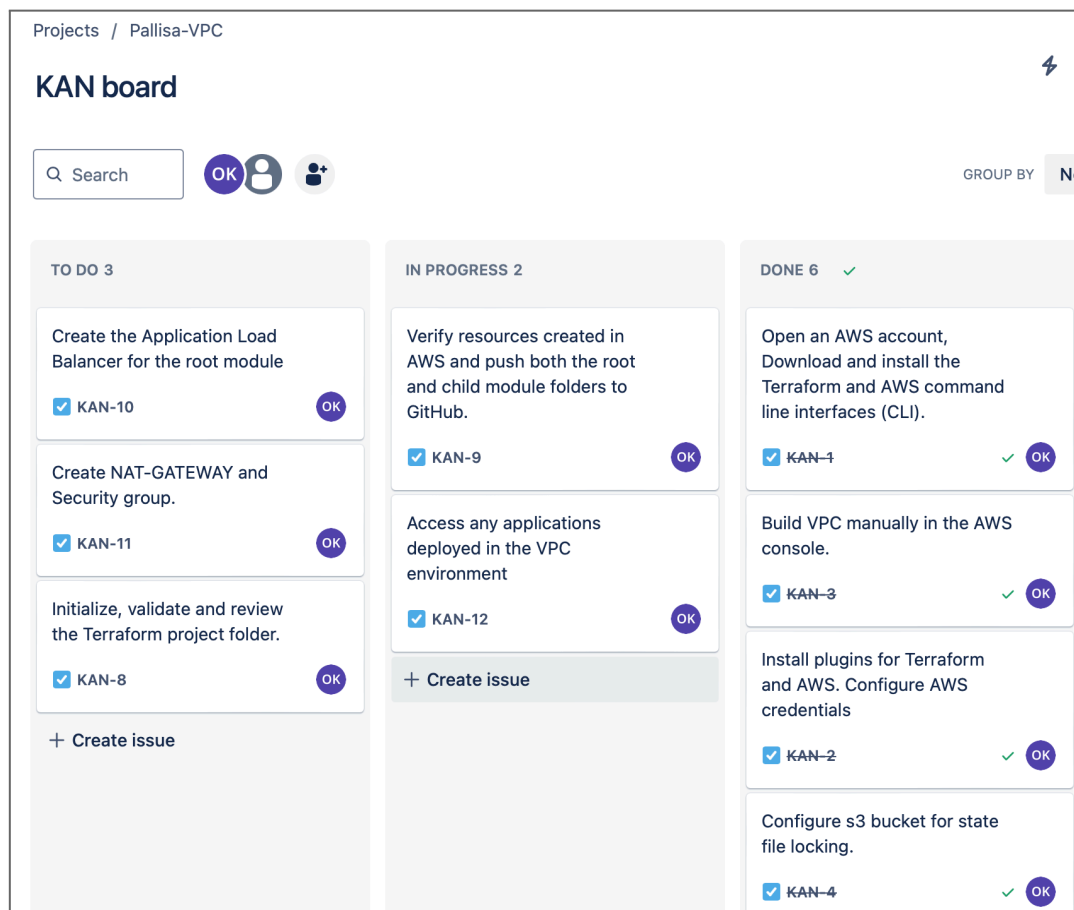
## 6 Implementation

### 6.1 Iteration 1

#### 6.1.1 Planning

In this chapter, the tools, plugins, and configurations necessary to build the VPC environment will be downloaded and set up, as shown in the [table of requirements](#) above, from ID 1 to 2.4.

In addition, the Kanban board below was created in Jira software for good visual management of the project.



*The screenshot above shows part of the Kanban board used for the project; created in Jira software.*

## 6.1.2 Implementation

As evidenced in [6.2 Iteration 2](#) below, all necessary tools and plugins were installed as per the table of requirements.

Put here a screenshot of the Kanban board

## 6.2 Iteration 2

### 6.2.1 Planning

Terraform state files will be written, tested and executed on macOS as the host operating system.

Step 1: Installing necessary tools and plugins.

- Install Terraform Command Line Interface (CLI)
- Install VS Code Editor and HashiCorp Terraform plugin for Visual Studio Code
- Install AWS CLI
- Configure AWS Credentials using SSH terminal on macOS

### 6.2.2 Terraform Installation CLI

- Download the Terraform binary for macOS from "[Install | Terraform | HashiCorp Developer](#)" using the commands below;

**"brew tap hashicorp/tap"**

**"brew install hashicorp/tap/terraform"**

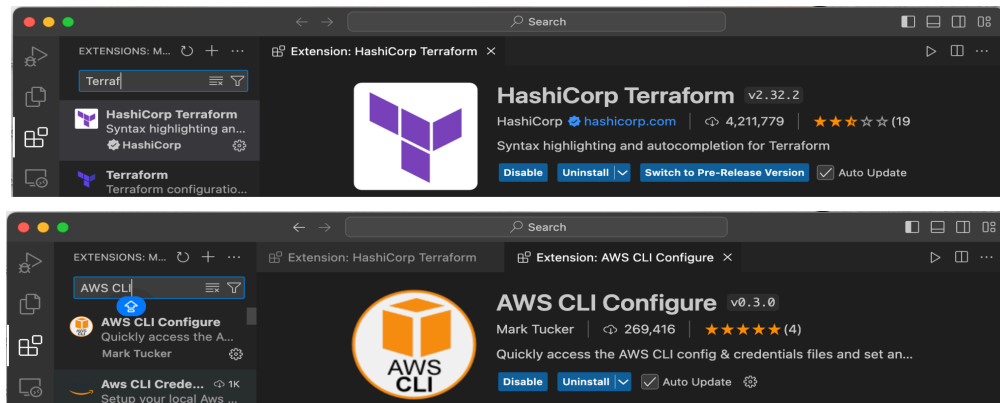
```
(base) otilliakawuma@Otillias-MacBook-Pro ~ % brew tap hashicorp/tap
[brew install hashicorp/tap/terraform
Running `brew update --auto-update`...
==> Downloading https://ghcr.io/v2/homebrew/portable-ruby/portable-ruby
#####
```

- Type the command **"terraform version"** to confirm the version of Terraform installed, which is "v1.9.5".

```
(base) otilliakawuma@Otillias-MacBook-Pro ~ % terraform version
Terraform v1.9.5
on darwin_arm64
(base) otilliakawuma@Otillias-MacBook-Pro ~ % █
```

### 6.2.3 Install the editor and its plugins

VS Code Editor, AWS\_CLI and Hashicorp terraform plugin for syntax high lighting and auto-completion in VS Code are installed as shown below.



## 6.2.4 Download and install the AWS\_CLI

The AWS-CLI was downloaded and configured from the official AWS site in my local working environment.

Command used;

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

The installed version and package location are shown as “2.15.24” and “/opt/homebrew/bin/aws,” respectively.

```
((base) otilliakawuma@Otillias-MacBook-Pro ~ % which aws
/opt/homebrew/bin/aws
((base) otilliakawuma@Otillias-MacBook-Pro ~ % aws --version
aws-cli/2.15.24 Python/3.11.8 Darwin/23.5.0 source/arm64 prompt/off
(base) otilliakawuma@Otillias-MacBook-Pro ~ %
```

## 6.2.5 Configure AWS CLI credentials

- An AWS user account is required for this stage.  
The account created includes 12 months of free tier access including the use of Amazon EC2, Amazon s3 and DynamoDB among others.
- Using the AWS Management Console, generate security credentials using services.
  - Go to Services → IAM → Users → “Your-Admin-User-Account” → Security Credentials  
The IAM user has programmatic access through the AWS-CL, they were added to an existing group with administrative access as AWS best practice.
  - Create Access Key
- Configure the created AWS credentials using the local terminal  
The command used; “aws configure”
  - Verify the AWS Credentials using the command “cat \$HOME/.aws/credentials”  
We have profiles; (kawuma-cli, the default and otillia-aws), we will be using [kawuma-cli]
  - Verify if we can list s3 buckets using the command “aws s3 ls”

```

● (base) otilliakawuma@Otillias-MacBook-Pro ~ % cat $HOME/.aws/credentials
[kawuma-cli]
aws_access_key_id = AKIAWJB5RPMZH4II43AR
aws_secret_access_key = KQeE1ESCMZyCbT2IxafoMLKc3Xd/Qzo141aQcY7
[otillia-aws]
aws_access_key_id = AKIAWJB5RPMZPIM74I3Z
aws_secret_access_key = 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
[default]
aws_access_key_id = AKIAWJB5RPMZH4II43AR
aws_secret_access_key = KQeE1ESCMZyCbT2IxafoMLKc3Xd/Qzo141aQcY7
● (base) otilliakawuma@Otillias-MacBook-Pro ~ % aws s3 ls
2023-09-01 02:20:25 applesch-website
2023-11-13 21:40:16 elasticbeanstalk-eu-west-2-431773285170
○ (base) otilliakawuma@Otillias-MacBook-Pro ~ % █

```

We now have permission to create AWS resources from the command line interface.

## 6.3 Iteration 3

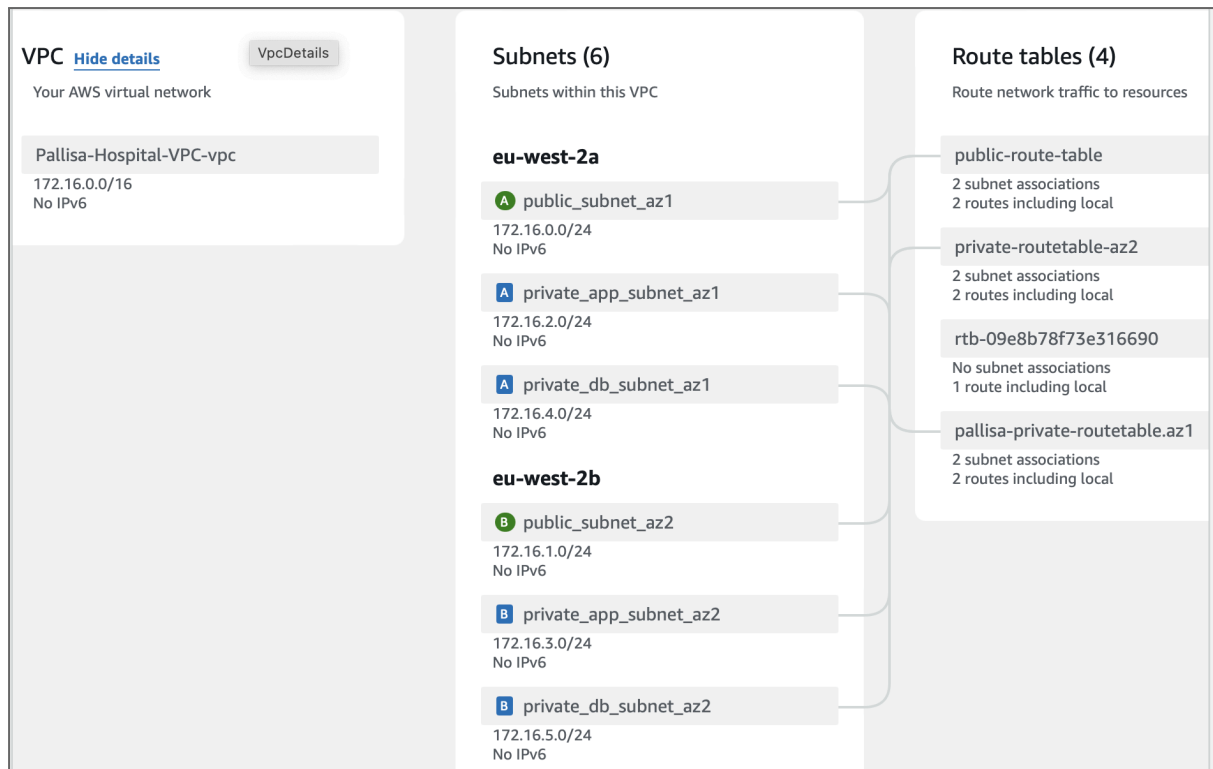
### 6.3.1 Planning

A decision was made to first create all core components of the VPC using the AWS console, this was tedious and time-consuming, but it allowed for a better understanding of the architecture components and their interactions. The hands-on experience provided valuable insight into needed for automating the process with Terraform.

### 6.3.2 Implementation

Console access to create VPC manually according to the [table of requirements](#) above.

1. On a browser, open the AWS Management Console, sign into your AWS account and navigate to the VPC Dashboard.
2. Initiate the VPC creation process by selecting the VPC option with public and private subnets across two AZs.
3. Create the internet gateway, to route traffic from resources in the public subnets
4. Create one public subnet and two private ones, ensuring they are distributed across two AZs for redundancy.
5. Create two route tables that control traffic flow within the VPC, associate the public subnet to the default main route table, and the application private subnets to the private route table that only connects to resources in the
6. Configure security groups, NAT gateways, and IAM policies to enforce security and access control in the subnets.



6.3.1 Visualization of the VPC created using the Amazon console.

## 6.4 Iteration 4

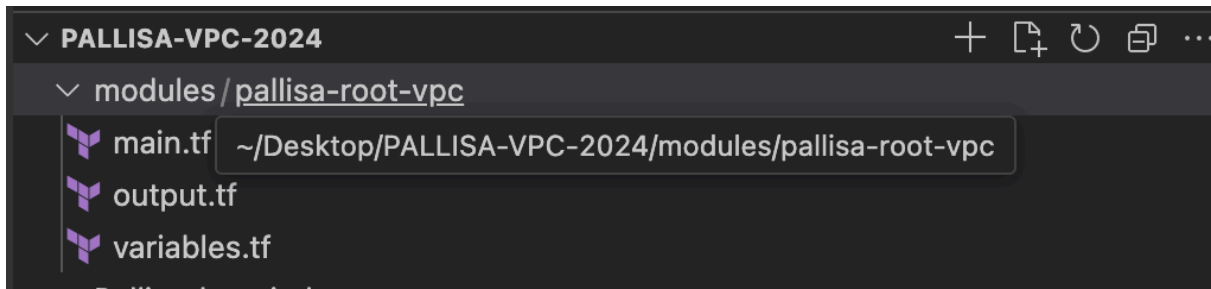
### 6.4.1 Planning

- A root VPC module folder will be created using a known module file from the Terraform registry. (*Terraform Registry*, 2024)
- The input variables for the VPC module will be created and referenced in the Pallisa Hospital VPC folder in [iteration 6.5](#).

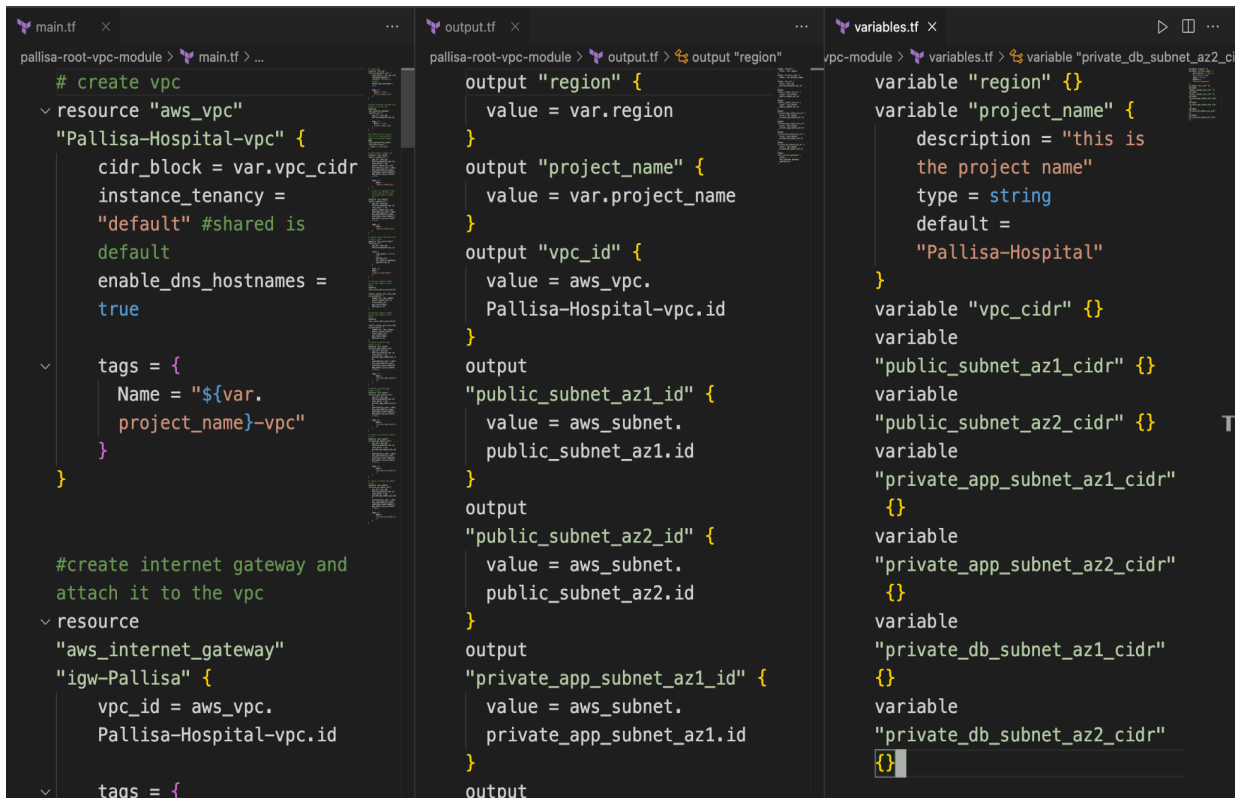
The module files above will allow us to create as many VPC as we like, but for this project, we will only use it to create a VPC for Pallisa General Hospital

### 6.4.2 Implementation

- After creating a new project folder, make a VPC folder to store all modules.
- Inside this root folder, we will touch the following Terraform files:
  - **main.tf**: will contain the main Terraform code of the VPC root module
  - **variable.tf**: will contain all defined variables
  - **outputs.tf**: will be used to export values referenced in the Hospital VPC



Screenshot showing the contents of the root module folder inside the project folder



The screenshot shows the contents of the **main.tf**, **output.tf** and **variables.tf**.

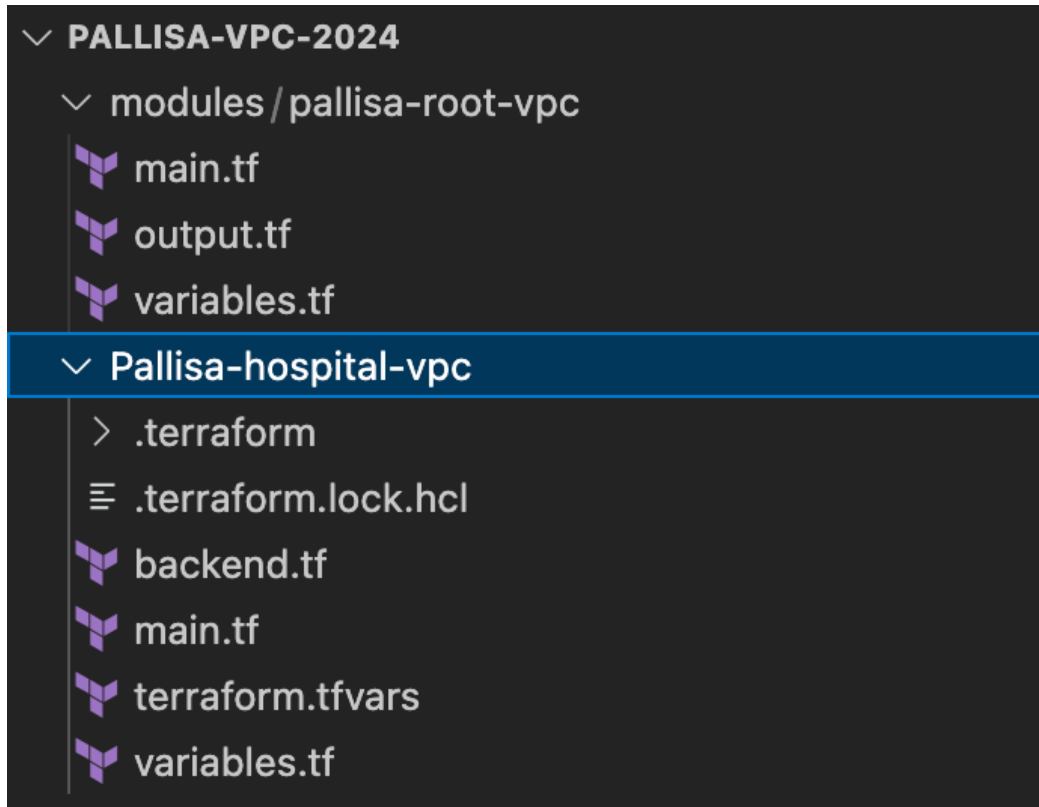
## 6.5 Iteration 5

### 6.5.1 Planning

We are now ready to use the root VPC module file created in [iteration 6.4](#) to create Pallisa Hospital's VPC infrastructure.

- Create a folder that will house the child module for Pallisa Hospital's VPC
- Create a s3 bucket for remote state storage in the AWS management console and also include the bucket name, key, region, and profile in the **"backend.tf"** file.
- Create the following files as well
  - **main.ft** : will contain code specific for Pallisa-Hospital-VPC
  - **variables.tf** : contains only variables needed for the VPC above

- **terraform.tfvars** : has local values for the variables identified in the **variables.tf** file above.



**Screenshot showing the main root module and child module folders separately created**

### 6.5.2 Testing the Terraform code

Here, we initialize the Terraform folder and files written in [iteration 5](#) above.

- Run **“terraform init”** to configure the s3 bucket, and root module and download the necessary plugins for AWS.



The screenshot shows a VS Code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project structure for 'PALLISA-VPC-2024' with subfolders 'modules' and 'Pallisa-hospital-vpc'. Inside 'Pallisa-hospital-vpc', there are files: '.terraform', '.terraform.lock.hcl', 'backend.tf', 'main.tf' (selected), 'terraform.tfvars', and 'variables.tf'. The 'main.tf' file is open in the editor, showing Terraform code for configuring the AWS provider and creating a VPC module. The terminal at the bottom shows the output of the 'terraform init' command, indicating that the backend is initialized and the provider plugins are installed.

```
1 # configure aws provider
2 provider "aws" {
3     region = var.region
4     profile = "kawuma-cli"
5 }
6
7 # create vpc for pallisa hospital using the vpc root module.
8 # reference the root vpc module
9 module "pallisa-hospital-vpc" {
```

```
(base) otilliakawuma@Otillias-MacBook-Pro Pallisa-hospital-vpc % terraform init
Initializing the backend...

Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing modules...
- pallisa-hospital-vpc in ../modules/pallisa-root-vpc-module
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.65.0...
- Installed hashicorp/aws v5.65.0 (signed by HashiCorp)
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!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
(base) otilliakawuma@Otillias-MacBook-Pro Pallisa-hospital-vpc %
```

The screen shot above shows that a custom VPC module initialized and used in creating a virtual private cloud for Pallisa Hospital.

- Run **terraform format "fmt"** and **"terraform validate"** to check syntax, and ensure code and configuration consistency respectively.

The screenshot shows a terminal window with the output of two Terraform commands. The first command is 'terraform fmt', which formats the files 'main.tf' and 'variables.tf'. The second command is 'terraform validate', which returns a success message indicating that the configuration is valid.

```
(base) otilliakawuma@Otillias-MacBook-Pro Pallisa-hospital-vpc % terraform fmt
main.tf
variables.tf
(base) otilliakawuma@Otillias-MacBook-Pro Pallisa-hospital-vpc % terraform validate
Success! The configuration is valid.
(base) otilliakawuma@Otillias-MacBook-Pro Pallisa-hospital-vpc %
```

- Run **"terraform plan"** to preview and verify infrastructure code  
This also provides a detailed document of the infrastructure to be provisioned.

```

+ cidr_block              = "172.16.1.0/24"
+ enable_dns64            = false
+ enable_resource_name_dns_a_record_on_launch = false
+ enable_resource_name_dns_aaaa_record_on_launch = false
+ id                      = (known after apply)
+ ipv6_cidr_block_association_id = (known after apply)
+ ipv6_native             = false
+ map_public_ip_on_launch = true
+ owner_id                = (known after apply)
+ private_dns_hostname_type_on_launch = (known after apply)
+ tags                    = {
+   + "Name" = "public_subnet_az2"
+ }
+ tags_all                 = {
+   + "Name" = "public_subnet_az2"
+ }
+ vpc_id                  = (known after apply)
}

# module.pallisa-hospital-vpc.aws_vpc.Pallisa-Hospital-vpc will be created
+ resource "aws_vpc" "Pallisa-Hospital-vpc" {
+   arn              = (known after apply)
+   cidr_block       = "172.16.0.0/16"
+   default_network_acl_id = (known after apply)
+   default_route_table_id = (known after apply)
+   default_security_group_id = (known after apply)
+   dhcp_options_id = (known after apply)
+   enable_dns_hostnames = true
+   enable_dns_support = true
+   enable_network_address_usage_metrics = (known after apply)
+   id                = (known after apply)
+   instance_tenancy = "default"
+   ipv6_association_id = (known after apply)
+   ipv6_cidr_block = (known after apply)
+   ipv6_cidr_block_network_border_group = (known after apply)
+   main_route_table_id = (known after apply)
+   owner_id           = (known after apply)
+   tags               = {
+     + "Name" = "Pallisa-Hospital-VPC-vpc"
+   }
+   tags_all           = {
+     + "Name" = "Pallisa-Hospital-VPC-vpc"
+   }
+ }

```

Plan: 11 to add, 0 to change, 0 to destroy.

- When happy with the plan above, run **“terraform apply”** to create the VPC in AWS cloud. This also updates the terraform state file and shows details of the provisioned infrastructure.

```

module.pallisa-hospital-vpc.aws_subnet.public_subnet_az1: Still creating... [10s elapsed]
module.pallisa-hospital-vpc.aws_subnet.public_subnet_az1: Creation complete after 11s [id=subnet-03cedf99bf72ef4b3]
module.pallisa-hospital-vpc.aws_route_table_association.public_subnet_az1_route_table_association: Creating...
module.pallisa-hospital-vpc.aws_subnet.public_subnet_az2: Creation complete after 11s [id=subnet-0934da084c3308fdb]
module.pallisa-hospital-vpc.aws_route_table_association.public_subnet_az2_route_table_association: Creating...
module.pallisa-hospital-vpc.aws_route_table_association.public_subnet_az1_route_table_association: Creation complete
module.pallisa-hospital-vpc.aws_route_table_association.public_subnet_az2_route_table_association: Creation complete

Apply complete! Resources: 11 added, 0 changed, 0 destroyed.
○ (base) otilliakawuma@Otillias-MacBook-Pro Pallisa-hospital-vpc %

```

### 6.5.2.1 Some errors encountered in code execution

```

○ (base) otilliakawuma@Otillias-MacBook-Pro PALLISA-VPC-2024 % pwd
/Users/otilliakawuma/Desktop/PALLISA-VPC-2024
○ (base) otilliakawuma@Otillias-MacBook-Pro PALLISA-VPC-2024 % cd Pallisa-hospital-vpc
○ (base) otilliakawuma@Otillias-MacBook-Pro Pallisa-hospital-vpc % terraform plan

Error: Module not installed

on main.tf line 9:
9: module "pallisa-hospital-vpc" {

This module's local cache directory ../modules/pallisa-root-vpc-module could not be read. Run "terraform init" to install all modules required by this configuration.

○ (base) otilliakawuma@Otillias-MacBook-Pro Pallisa-hospital-vpc %




```

The above error is a confirmation that the code structure is configured to use the custom Pallisa-VPC module to build infrastructure in AWS, without it, nothing can be created, the module needs to be downloaded and used.








This was resolved by providing the correct module correct location path within the project folder. When the “terraform init” command was run as advised in the error, the VPC module was downloaded and initialized for resource provisioning as shown above in [6.5.2. Testing the Terraform code.](#)

## 6.5 Review



6.5.1 Below is the VPC created in Amazon Web Services using terraform code.

vpc-0ccd6f37bdea7ca9d / Pallisa-Hospital-VPC-vpc				Actions
Details Info				
VpcDetails				
VPC ID	State	DNS hostnames	DNS resolution	
 vpc-0ccd6f37bdea7ca9d	 Available	Enabled	Enabled	
Tenancy	DHCP option set	Main route table	Main network ACL	
Default	dopt-0e18ac0a833c2ecb2	rtb-06bac699ea115349f	acl-040ca2ec8174bd916	
Default VPC	IPv4 CIDR	IPv6 pool	IPv6 CIDR (Network border group)	
No	172.16.0.0/16	-	-	
Network Address Usage metrics	Route 53 Resolver DNS Firewall rule groups	Owner ID		
Disabled	-	 431773285170		

The VPC is isolated into two public subnets for public access and four private subnets for private and secure access within the VPC, as shown below.

<input type="checkbox"/>	private_app_subnet_az1	<a href="#">subnet-0f3e0248bca3c6271</a>	 Available.	<a href="#">vpc-09988e0771a08b135   Pallisa-Hospital-VPC-vpc</a>
<input type="checkbox"/>	private_app_subnet_az2	<a href="#">subnet-0199fad99ae8a59f2</a>	 Available.	<a href="#">vpc-09988e0771a08b135   Pallisa-Hospital-VPC-vpc</a>
<input type="checkbox"/>	private_db_subnet_az1	<a href="#">subnet-03e22414d5a56dd34</a>	 Available.	<a href="#">vpc-09988e0771a08b135   Pallisa-Hospital-VPC-vpc</a>
<input type="checkbox"/>	private_db_subnet_az2	<a href="#">subnet-0a321a616e2821d3f</a>	 Available.	<a href="#">vpc-09988e0771a08b135   Pallisa-Hospital-VPC-vpc</a>
<input type="checkbox"/>	public_subnet_az1	<a href="#">subnet-0133b1ee7a53e9acd</a>	 Available.	<a href="#">vpc-09988e0771a08b135   Pallisa-Hospital-VPC-vpc</a>
<input type="checkbox"/>	public_subnet_az1	<a href="#">subnet-0de597e377a07ea41</a>	 Available.	<a href="#">vpc-0ccd6f37bdea7ca9d   Pallisa-Hospital-VPC-vpc</a>
<input type="checkbox"/>	public_subnet_az2	<a href="#">subnet-0c584688460b01ee2</a>	 Available.	<a href="#">vpc-09988e0771a08b135   Pallisa-Hospital-VPC-vpc</a>

Internet gateway, allowing both inbound and outbound traffic from the internet as per the VPC Architecture in [5.3 VPC Architecture Overview](#) above.

igw-0d07716d2b792bc9f / Pallisa-Hospital-VPC-igw		
Details Info		
Internet gateway ID	State	VPC ID
 igw-0d07716d2b792bc9f	 Attached	<a href="#">vpc-09988e0771a08b135   Pallisa-Hospital-VPC-vpc</a>

## Public Route for Public Subnets access

<input type="checkbox"/>	public_subnet_az1	<a href="#">subnet-0133b1ee7a5...</a>	172.16.0.0/24	-	<a href="#">rtb-072ca04b4f91a2e71 / public-route-table</a>
<input type="checkbox"/>	public_subnet_az2	<a href="#">subnet-0c584688460...</a>	172.16.1.0/24	-	<a href="#">rtb-072ca04b4f91a2e71 / public-route-table</a>

There's no public route for the private subnets below, we instead have private route tables as shown in 6.3.1 Visualization of the VPC created using the Amazon console above.

Explicit subnet associations (2)			
<input type="text" value="Find subnet association"/>			
Name	Subnet ID	IPv4 CIDR	
private_app_subnet_az1	<a href="#">subnet-0f3e0248bca3c6271</a>	172.16.2.0/24	RouteTables
private_db_subnet_az1	<a href="#">subnet-03e22414d5a56dd34</a>	172.16.4.0/24	

Network Address Translation (NAT) Gateway only allows private network resources to connect to the internet, it prevents internet users from initiating connections with resources.

<input type="radio"/>	pallisa-nat-gateway-az1	<a href="#">nat-00324e8c2f00509d0</a>	Public	✓ Available
<input type="radio"/>	pallisa-nat-gateway-az2	<a href="#">nat-0c00313596db74fb8</a>	Public	✓ Available

Security groups created for application load balancers and elastic container services, controlling inbound and outbound traffic for resources in the app and database tiers of the VPC.

<input type="checkbox"/>	alb-security-group	<a href="#">sg-07e967abf95fe916d</a>	alb-security-group
<input type="checkbox"/>	ecs-security-group	<a href="#">sg-0786ee00ee517991f</a>	ecs-security-group

## 6.5.1 Terraform destroy

For cost management, the reviewed resources above were successfully destroyed using the “**terraform destroy**” command as shown below.

```
module.Nat-gateway.aws_eip.eip_for_nat_gateway_az1: Destruction complete after 1s
module.Nat-gateway.aws_eip.eip_for_nat_gateway_az2: Destruction complete after 1s

Destroy complete! Resources: 26 destroyed.
(base) otilliakawuma@Otillias-MacBook-Pro Pallisa-hospital-vpc %
```

# 7 Evaluation

## 7.1 Table of Requirements

To ensure the VPC meets the desired objectives set out in [1.3 Project Objectives](#), we will use our [4.4 Table of Requirements](#) to evaluate the infrastructure created in [iterations 6.4](#) and [6.5](#).

## 4.4 Table of Requirements

ID	Description	Status	Explanation
1	Open an AWS account	Satisfied ▾	Necessary to perform API calls in the AWS cloud
1.1	Knowledge of AWS services and how Terraform works	Satisfied ▾	Faster coding and a good understanding of terraform
2	Download and install the Terraform command line interface (CLI)	Satisfied ▾	Quick access to Terraform resources was successful
2.1	Download and install the AWS command line interface	Satisfied ▾	Quick access to CLI and credential files was obtained
2.3	Install plugins for HashiCorp Terraform, AWS CLI configure,	Satisfied ▾	Syntax auto-completion <a href="#">6.2.3 Install the editor and its plugins</a>
2.4	Configure AWS credentials	Satisfied ▾	Access rights and credentials working well
3	Build VPC manually in the AWS console	Satisfied ▾	Understand console interaction of resources in the AWS cloud
4	Configure s3 bucket for state file locking	Satisfied ▾	For terraform state storage and locking
5	Write the Terraform VPC modules	Satisfied ▾	For re-usability <a href="#">6.4.2 Implementation</a>
5.1	Configure the project variables	Satisfied ▾	For a specific use case
5.2	Initialize, validate and review the Terraform project folder	Satisfied ▾	As discussed in <a href="#">6.5.2 Testing the Terraform code</a>
5.4	Review VPC infrastructure is running	Satisfied ▾	Network traffic isolated into subnets as shown in <a href="#">6.5.3 Review</a>

5.6	Access any applications deployed in the VPC environment	Partial... ▾	Further testing and verification needed
6	Push project details to GitHub	Satisfied ▾	All repositories were successfully pushed

## 7.2 Limitations in using IaC tools like Terraform

Terraform, a robust tool for infrastructure management, comes with notable limitations. It may lag in supporting new cloud provider features, often requiring workarounds for complex configurations. This can increase maintenance overhead. Additionally, Terraform presents a steep learning curve, demanding mastery of its specific HCL syntax and understanding of complex concepts like state management and resource dependencies. Organizations with strict compliance requirements may face challenges in securely managing Terraform state files, which typically contain sensitive resource information. Despite these constraints, Terraform remains valuable for infrastructure-as-code implementations. Users should be aware of these limitations to effectively plan and execute their infrastructure strategies, balancing Terraform's powerful capabilities with its potential drawbacks.

### 7.2.1 Key Limitations of Building Infrastructure in AWS

1. VPC and Subnet Limits:
  - VPCs per Region: By default, you can have up to 5 VPCs per region, but this limit can be increased by contacting AWS support.
  - Subnets per VPC: Each VPC can have up to 200 subnets, which is generally sufficient for most use cases but can be increased if needed.
2. CIDR Blocks:
  - IPv4 CIDR Blocks: Each VPC can have up to 5 IPv4 CIDR blocks, including the primary and secondary blocks. This limit can be increased to a maximum of 50 upon clearance from AWS at a fee.
3. IPv6 CIDR Blocks: The default limit is 1 IPv6 CIDR block per VPC, which cannot be increased.
4. Security Groups and Network ACLs:
  - Security Groups per Region: The default limit is 2,500 security groups per region, which can be increased. Each security group can have up to 60 inbound and 60 outbound rules.
  - Network ACLs per VPC: You can have up to 200 network ACLs per VPC, with each ACL supporting up to 20 rules.
5. Elastic IP Addresses:
  - Elastic IPs per Region: The default limit is 5 Elastic IP addresses per region, which can be increased if necessary.

#### 6. Gateways and Peering Connections:

- Internet and NAT Gateways: Each region can have up to 5 internet gateways and 5 NAT gateways per availability zone.
- VPC Peering Connections: A VPC can have up to 125 active peering connections, with a limit of 25 outstanding requests

#### 7. Flow Logs:

- Flow Logs per Resource: Each network interface, subnet, or VPC can have up to 2 flow logs, which cannot be increased.

### 7.2.2 Mitigating Limitations

- Requesting Increases: Most of the limits are soft limits and can be increased by submitting a request to AWS support through the AWS Management Console.
- Optimizing Resource Usage: To efficiently manage and optimize the use of resources, such as CIDR blocks and security groups, stay within set limits.
- Planning for Growth: Consider future scalability needs when designing VPC architecture, and plan for potential increases in resource limits.

Understanding these limitations is crucial for effectively designing and managing VPCs on AWS, ensuring that the infrastructure can easily scale and adapt to organizational needs.

## 8 Conclusion

The execution of a three tier Virtual Private Cloud (VPC) for Pallisa Hospital using Terraform in AWS provided a robust and scalable infrastructure that can support the hospital's IT needs.

Key attainments of this proposal entail a custom VPC module file that is easily deployable, the possibility of remote working, electronic management of patient data, enhanced collaboration between IT and other hospital staff and growth opportunities in telemedicine.

### 8.2 Future Work

1. Continuous Improvement and Optimization:
  - Regularly review and update the VPC architecture to incorporate new AWS features and services that enhance performance, security, and cost-efficiency.
2. Enhanced Security Measures:
  - Explore the integration of additional security services, such as AWS Shield for DDoS protection, to further strengthen the security posture of the VPC.
  - Conduct regular security audits and penetration testing to identify and mitigate potential vulnerabilities.
3. Scalability and Expansion:
  - Plan for future scalability by evaluating the potential need for additional resources, such as increased IP address space connections to accommodate growth in hospital operations such as remote working, as mentioned in
4. Training and Capacity Building:
  - Provide ongoing training for IT staff on the latest cloud technologies and best practices to ensure they are equipped to manage and optimize the VPC infrastructure effectively.
  - Foster a culture of continuous learning and development within the hospital's IT team to drive future improvements and digital transformation initiatives.



# References

- Agile Software Development Methodologies: Survey of Surveys. (2017, March). *Agile Software Development Methodologies: Survey of Surveys*, 5(5).  
10.4236/jcc.2017.55007
- Anthony, A. (2017). *Mastering AWS Security*. Packt Publishing. [www.packtpub.com](http://www.packtpub.com)
- Bhatia, S., & Gabhane, C. (2023). *Reverse Engineering with Terraform: An Introduction to Infrastructure Automation, Integration, and Scalability Using Terraform*. Apress.  
<https://link.springer.com/book/10.1007/979-8-8688-0074-0>
- The Future of Cloud Computing*. (n.d.). Private cloud. Retrieved June 14, 2024, from [www.scirp.org](http://www.scirp.org)
- Hashicorp Inc. (2018, March). *Terraform Home*. Terraform Home. Retrieved August 28, 2024, from <https://developer.hashicorp.com/terraform/intro>
- Hiwarkar, K., Doshi, A., Chinta, R., & Manjula, R. (2016). *Comparative Analysis of Agile Software Development Methodologies*.  
<https://api.semanticscholar.org/CorpusID:33559437>. Retrieved 09 07, 2024, from <https://api.semanticscholar.org/CorpusID:33559437>
- Ivanova, D. (DECEMBER 10 2018). Development of PaaS using AWS and Terraform for medical imaging analytics. *Development of PaaS using AWS and Terraform for medical imaging analytics*, (060018), 060018-1. 10.1063/1.5082133
- Jayaraman, S. (2024, June 6). *160+ Fascinating Cloud Computing Statistics for 2024*. G2. Retrieved June 13, 2024, from <https://www.g2.com/articles/cloud-computing-statistics>
- Kavas, E. (2023). *Architecting AWS with Terraform: Design Resilient and Secure Cloud Infrastructures with Terraform on Amazon Web Services*. Packt Publishing.

Leveraging Terraform as a Battle-Tested Solution. (2024, May 21). *Streamlining Multi-Cloud Infrastructure Orchestration: Leveraging Terraform as a Battle-Tested Solution*, 911-915. 10.1109/ICC-ROBINS60238.2024.10533995

*Local Government Performance Assessment*. (n.d.). Uganda Budget Information. Retrieved August 19, 2024, from [https://budget.finance.go.ug/sites/default/files/pallisa\\_district\\_assessment\\_lgpa\\_full\\_report-compressed.pdf](https://budget.finance.go.ug/sites/default/files/pallisa_district_assessment_lgpa_full_report-compressed.pdf)

Mehdi, A., & Walia, R. (2024, February 14). Streamlining Infrastructure Deployment and Management Through Infrastructure as Code. *Streamlining Infrastructure Deployment and Management Through Infrastructure as Code*. 10.1109/ICCCIS60361.2023.10425616

Muthoni, S. S., Okeyo, G. G., & Chemwa, G. G. (2021, December 09). 2021 International Conference on Electrical, Computer and Energy Technologies (ICECET). *Infrastructure as code for business continuity in institutions of higher learning*. 10.1109/icecet52533.2021.9698544

Odeyinka, O. J. (2024). *AWS Cloud Automation: In-depth Guide to Automation Using Terraform Infrastructure as Code Solutions (English Edition)*. Bpb Publications.

OpenAI. (2024, 06 13). *ChatGPT*. Virtual Cloud Implementation Project. <https://chatgpt.com/share/a25b9df6-becb-417f-b0ae-ffaff4e1c7dc>

Tak, A. (2023, May 25). *Journal of Artificial Intelligence & Cloud Computing, Volume 2(2): 1-7*. [www.onlinescientificresearch.com](http://www.onlinescientificresearch.com)

*Terraform Registry*. (n.d.). Terraform Registry. Retrieved August 27, 2024, from <https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance>

*Terraform Registry*. (2024). HashiCorp. Retrieved 08 27, 2024, from <https://registry.terraform.io/providers/hashicorp/aws>

Tetteh, S. G. (2024, 02 27). Empirical Study of Agile Software Development Methodologies: A Comparative Analysis. *Asian Journal of Research in Computer Science*, 17(5), 30-35. 10.9734/AJRCOS/2024/v17i5436

What is Terraform. (n.d.). What is Terraform. Retrieved June 14, 2024, from  
[www.terraform.io/](http://www.terraform.io/)

## Appendix I — Certificate of Ethics Review

**Project title:** Proposed Virtual Private Cloud for Pallisa Hospital. Provisioned in Amazon Web Services (AWS) using terraform.

<b>Name:</b>	Otillia Kawuma	<b>User ID:</b>	2065012 <b>Application date:</b> 31/05/2024 <b>ER Number:</b> 15:29:12	TETHIC-2024-108847
--------------	----------------	-----------------	--	--------------------

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are [Elisavet Andrikopoulou](#), [Kirsten Smith](#)

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- [University Policy](#)
- [Safety on Geological Fieldwork](#)

It is also your responsibility to follow University guidance on Data Protection Policy:

- [General guidance for all data protection issues](#)
- [University Data Protection Policy](#)

Which school/department do you belong to?: **School of Computing**

What is your primary role at the University?: **Postgraduate Student**

What is the name of the member of staff who is responsible for supervising your project?: **Mani Ghahremani** Is the study likely to involve human subjects (observation) or participants?: No

Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No

Are there risks of significant damage to physical and/or ecological environmental features?: No

Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No

Does the project involve animals in any way?: No

Could the research outputs potentially be harmful to third parties?: No

Could your research/artefact be adapted and be misused?: No

Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: No

Please read and confirm that you agree with the following statements: I confirm that I have considered the implications for data collection and use, taking into consideration legal requirements (UK GDPR, Data Protection Act 2018 etc.), I confirm that I have considered the impact of this work and and taken any reasonable action to mitigate potential misuse of the project outputs, I confirm that I will act ethically and honestly throughout this project

## Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University

Ethics Policy. Supervisor comments:

Supervisor's Digital Signature: mani.ghahremani@port.ac.uk Date: 05/07/2024

## Appendix II — Sample code of Pallisa Hospital's VPC.

### main.tf of root module

```
# create vpc
resource "aws_vpc" "Pallisa-Hospital-vpc" {
  cidr_block = var.vpc_cidr
  instance_tenancy = "default" #shared is default
  enable_dns_hostnames = true

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

#create internet gateway and attach it to the vpc
resource "aws_internet_gateway" "igw-Pallisa" {
  vpc_id = aws_vpc.Pallisa-Hospital-vpc.id

  tags = {
    Name = "${var.project_name}-igw"
  }
}

#use data source to get a list of all availability zones in preferred region
data "aws_availability_zones" "available_zones" {
  state = "available"
}

#create public subnet az1
resource "aws_subnet" "public_subnet_az1" {
  vpc_id = aws_vpc.Pallisa-Hospital-vpc.id
  cidr_block = var.public_subnet_az1_cidr
  availability_zone = data.aws_availability_zones.available_zones.names[0]
  map_public_ip_on_launch = true

  tags = {
    Name = "public_subnet_az1"
  }
}
```

```

    # az2 is indexed to the first zone by the zero.
    # create public subnet az2
resource "aws_subnet" "public_subnet_az2" {
    vpc_id = aws_vpc.Pallisa-Hospital-vpc.id
    cidr_block = var.public_subnet_az2_cidr
    availability_zone = data.aws_availability_zones.available_zones.names[1]
    map_public_ip_on_launch = true

    tags = {
        Name = "public_subnet_az2"
    }
}

# create route table and add a public route
resource "aws_route_table" "Pallisa-rt" {
    vpc_id = aws_vpc.Pallisa-Hospital-vpc.id

    route {
        cidr_block = "0.0.0.0/0"
        gateway_id = aws_internet_gateway.igw-Pallisa.id
    }

    tags = {
        Name = "public-route-table"
    }
}

# associate public subnet az1 to the "public route table"
resource "aws_route_table_association" "public_subnet_az1_route_table_association" {
    subnet_id = aws_subnet.public_subnet_az1.id
    route_table_id = aws_route_table.Pallisa-rt.id
}

# associate public subnet az2 to the "public route table"
resource "aws_route_table_association" "public_subnet_az2_route_table_association" {
    subnet_id = aws_subnet.public_subnet_az2.id
    route_table_id = aws_route_table.Pallisa-rt.id
}

# create a private app subnet in az1
resource "aws_subnet" "private_app_subnet_az1"{
    vpc_id = aws_vpc.Pallisa-Hospital-vpc.id
    cidr_block = var.private_app_subnet_az1_cidr
    availability_zone = data.aws_availability_zones.available_zones.names[0]
    map_public_ip_on_launch = false

    tags = {
        Name = "private_app_subnet_az1"
    }
}

```

```

}

# create a private app subnet in az2
resource "aws_subnet" "private_app_subnet_az2"{
  vpc_id = aws_vpc.Pallisa-Hospital-vpc.id
  cidr_block = var.private_app_subnet_az2_cidr
  availability_zone = data.aws_availability_zones.available_zones.names[1]
  map_public_ip_on_launch = false

  tags = {
    Name = "private_app_subnet_az2"
  }
}

# create a private db subnet in az1
resource "aws_subnet" "private_db_subnet_az1"{
  vpc_id = aws_vpc.Pallisa-Hospital-vpc.id
  cidr_block = var.private_db_subnet_az1_cidr
  availability_zone = data.aws_availability_zones.available_zones.names[0]
  map_public_ip_on_launch = false

  tags = {
    Name = "private_db_subnet_az1"
  }
}

# create a private db subnet in az2
resource "aws_subnet" "private_db_subnet_az2"{
  vpc_id = aws_vpc.Pallisa-Hospital-vpc.id
  cidr_block = var.private_db_subnet_az2_cidr
  availability_zone = data.aws_availability_zones.available_zones.names[1]
  map_public_ip_on_launch = false

  tags = {
    Name = "private_db_subnet_az2"
  }
}

```

## main.tf file of child module

```

# configure aws provider
provider "aws" {
  region = var.region
  profile = "kawuma-cli"
}

# create vpc for pallisa hospital using the vpc root module.

```

```

# reference the root vpc module
module "pallisa-hospital-vpc" {
  source          = "../modules/pallisa-root-vpc"
  region          = var.region
  project_name    = var.project_name
  vpc_cidr        = var.vpc_cidr
  public_subnet_az1_cidr = var.public_subnet_az1_cidr
  public_subnet_az2_cidr = var.public_subnet_az2_cidr
  private_app_subnet_az1_cidr = var.private_app_subnet_az1_cidr
  private_app_subnet_az2_cidr = var.private_app_subnet_az2_cidr
  private_db_subnet_az1_cidr = var.private_db_subnet_az1_cidr
  private_db_subnet_az2_cidr = var.private_db_subnet_az2_cidr
}

# created Pallisa-Nat-gateway
module "Nat-gateway" {
  source          = "../modules/Nat-gateway"
  public_subnet_az1_id    = module.pallisa-hospital-vpc.public_subnet_az1_id
  aws_internet_gateway    = module.pallisa-hospital-vpc.aws_internet_gateway
  vpc_id                = module.pallisa-hospital-vpc.vpc_id
  public_subnet_az2_id    = module.pallisa-hospital-vpc.public_subnet_az2_id
  private_app_subnet_az1_id = module.pallisa-hospital-vpc.private_app_subnet_az1_id
  private_db_subnet_az1_id = module.pallisa-hospital-vpc.private_db_subnet_az1_id
  private_app_subnet_az2_id = module.pallisa-hospital-vpc.private_app_subnet_az2_id
  private_db_subnet_az2_id = module.pallisa-hospital-vpc.private_db_subnet_az2_id
}

# created security group
module "sec_group" {
  source = "../modules/sec-group"
  vpc_id = module.pallisa-hospital-vpc.vpc_id
}

# created application load balancer
module "app-load-balancer" {
  source          = "../modules/app-load-balancer"
  project_name    = module.pallisa-hospital-vpc.project_name
  alb_security_group_id = module.sec_group.alb_security_group_id
  public_subnet_az1_id = module.pallisa-hospital-vpc.private_db_subnet_az1_id
  public_subnet_az2_id = module.pallisa-hospital-vpc.public_subnet_az2_id
  vpc_id            = module.pallisa-hospital-vpc.vpc_id
}

```

**Link to GitHub repository for code in-depth viewing**

<https://github.com/Okawuma/Pallisa-Hospital-VPC2024.git>