OULUN AMMATTIKORKEAKOULU

Md Moniruzzaman

MERN STACK APPLICATION DEPLOYMENT IN THE CLOUD AND AUTOMATION PROCESS USING TERRAFORM

MERN STACK APPLICATION DEPLOYMENT IN THE CLOUD AND AUTOMATION PROCESS USING TERRAFORM

Md Moniruzzaman
Bachelor's Thesis
Autumn 2022
Bachelor of Engineering
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Bachelor of Engineering, Information Technology

Author(s): Md Moniruzzaman
Title of the thesis: MERN Stack Web Application Deployment in the Cloud and Automation Process Using Terraform
Thesis examiner(s): Teemu Korpela
Term and year of thesis completion: Autumn 2022 Pages: 41 + 1 appendix

This Bachelor's thesis aimed to provide the audience with a clear idea of deploying a MERN stack application in the cloud and automation process using Terraform. Maintaining cloud infrastructure of any substantial size can often require performing the same actions or procedures over and over. There is, however, another way; the proposed solution was to utilize Terraform to automate a well-defined and concise way to deploy infrastructure resources and changes.

The introduction chapter introduced the theme of the thesis. Cloud technologies used in this thesis were evaluated with their pros and cons. In this thesis, a very tiny fraction of AWS cloud technologies was introduced. The work started by developing a step-by-step process of deploying the MERN stack application using AWS cloud-native technologies. This thesis has profiled brief research and implementation of DevOps practices using HashiCrop Terraform. The concluding chapter summarises the previous chapters and envisages how future studies will help to develop the system in the upcoming days.

It was evident that a complete cloud environment would be necessary to achieve high-performance web applications. However, this could not be accomplished while using the manual deployment technique. By adopting Terraform Technologies, these issues will be solved, thus resulting in a perfect solution to increase overall productivity.

Keywords: AWS, Terraform, EC2, VPC, Nginx, CLI, Amazon Linux

PREFACE

This thesis was submitted for the degree of Information Technology at the Oulu University of Applied Sciences. The thesis described herein was conducted under Lecturer Teemu Korpela in the department of Information Technology, Oulu University of Applied Sciences, during the autumn of 2022.

From the Oulu University of Applied Sciences, I am grateful to my supervisor Teemu Korpela for his kind support & guidance during my thesis journey.

Oulu, 8 November 2022
Md Moniruzzaman

ABBREVIATIONS

API	Application Programming Interface
AWS	Amazon Web Services
CDN	Content Delivery Network
CI/CD	Continuous Integration and Continuous Delivery
CLI	Command-line Interface
DB	Database
DevOps	Development and Operations
EC2	Elastic Compute Cloud
GCP	Google Cloud Platform
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaC	Infrastructure as Code
IAM	AWS Identity and Access Management
ICPM	Internet Control Message Protocol
IG	Internet Gateway
IP	Internet Protocol
JSON	JavaScript Object Notation
MERN	MongoDB, Express, React, Node
NPM	Node Package Manager
SSH	Secure Shell
TCP	Transmission Control Protocol
UI	User Interface
VPC	Virtual Private Cloud
VPN	Virtual Private Network

CONTENTS

1 INTRODUCTION

Cloud computing is one of the hottest buzzwords in the 21st century and is considered the fastest-growing tech industry. The term “cloud computing” was first coined by *George Favaloro* in 1996 while writing his Compaq Business plan. In 1997, *Chellappa* provided the first academic definition of the phrase "cloud computing," and in 2007, the term began to gain traction. According to the National Institute of Standards and Technology (NIST), cloud computing is “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”(P. Mell and T. Grance, 2011).

Cloud-based products, such as storage, computation, databases, mobile developer tools, IoT, AI, enterprise apps, networking, management tools, and many more, are within short reach. These give enterprises, start-ups, and big businesses the perfect opportunity for easy adaptation in a rapidly changing business environment to be continuously successful.

AWS, Microsoft Azure, and Google Cloud are the key players in the cloud industry. According to Statista, Amazon leads the two-hundred-billion-dollar cloud market. Amazon has over 200+ available services. Amazon's market share is about 34% among cloud services providers; Microsoft Azure has 21%, followed by Google Cloud, which counts only 10% of the share. [3]

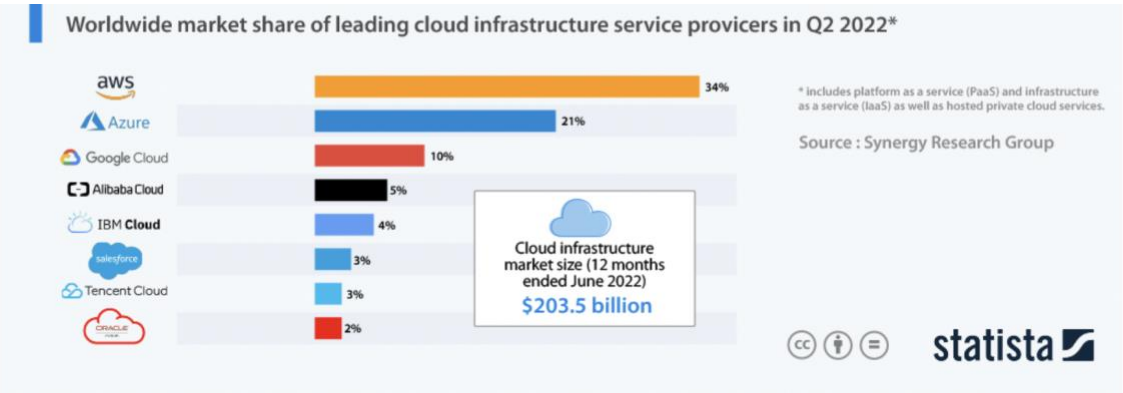


FIGURE 1. Worldwide market share of major cloud services providers. [3]

Terraform is an infrastructure as code (IaC) tool. IaC is the fundamental pillar of implementing DevOps practices. Terraform allows developers to configure and build local and remote instances. The use of IaC scripts helps developers to increase their deployment speed. [4]

Amazon Web Services (AWS) will be the foundation of my thesis because it accounts for around 34% of the existing public cloud services industry. Amazon Web Services (AWS) has established itself as a leader in cloud computing and a model for other cloud service providers.

This thesis aims to identify and study different AWS services & follow the best industry practices to implement them in real-life scenarios.

2 CLOUD COMPUTING MODELS

“Cloud Computing” is a newly emerged computing paradigm. Cloud Computing has been provided by prominent service providers like Amazon, IBM, Google, Oracle, Microsoft, and many more. The Cloud computing model, also known as CBMF (Cloud Business Model Framework), is mainly categorized into three layers: the infrastructure layer, the platform-as-a-service layer, and the application layer (SaaS) on top. This chapter will provide more profound information by providing insights into these categories.

2.1 Infrastructure as a Service (IaaS)

IaaS is a cloud computing model that allows cloud developers total control over their infrastructure without needing physical hardware. It offers on-demand cloud computing services via the internet, including networking, storage, and other infrastructure components. Cloud developers can configure their infrastructure according to their needs. IaaS is divided into three major segments: computation, network, and storage. It provides cloud developers with the building blocks to develop specialized systems. [5]

2.2 Platform as a Service (PaaS)

PaaS, or Platform-as-a-Service, is a cloud computing model that provides the cloud developer with a complete cloud platform. This way, a cloud developer can run applications without managing underlying infrastructure (for example, updates, maintenance, hardware, and operating systems). PaaS also provides a well-defined framework for cloud developers to create customized applications. [8]

2.3 Software as a Service (SaaS)

SaaS, or software-as-a-service, is cloud-based application software and easily accessible by a web browser or mobile application. The Cloud vendor oversees running, managing, and supporting the software and the infrastructure on which it runs. SaaS is one of the most prominent software delivery models nowadays. Many popular day-to-day utilities, including Gmail (for email), Dropbox (for file storage and sharing), and many essential corporate programs, are offered via the SaaS model. SaaS offers pre-built solutions that are beneficial to companies of all kinds. Organizations and Companies can grow quickly with minimal to no managerial overhead and a predictable cost model. [6]

2.4 Comparisons of Cloud Computing Models

IaaS, PaaS, and SaaS are the three most popular 'as a service' types of computing models offered by cloud service providers. Many enterprises use all three cloud models, often in combination with traditional IT. The main difference between those services is management ease versus complete control. TABLE 1 shows the different hardware and software components required to run an application. It illustrates how developers progressively manage fewer components as the company goes from the traditional on-premises computing service model to the SaaS model. It is evident that the on-premises cloud computing model, also known as private cloud, has hundred percent control over everything, including hardware and software, which is also discussed in section 3.2

TABLE 1. Management responsibilities for cloud computing models. [11]

Control over resources

→

	On-Premises	IaaS	PaaS	SaaS
Application	🟩	🟩	🟩	🟦
Data	🟩	🟩	🟩	🟦
Runtime	🟩	🟩	🟦	🟦
Middleware	🟩	🟩	🟦	🟦
OS	🟩	🟦	🟦	🟦
Virtualization	🟩	🟦	🟦	🟦
Servers	🟩	🟦	🟦	🟦
Storage	🟩	🟦	🟦	🟦
Networking	🟩	🟦	🟦	🟦

3 CLOUD DEPLOYMENTS MODELS

According to the NIST definition, there are four cloud computing deployment models: private clouds, public clouds, hybrid clouds, and community clouds represent the cloud environments in which a developer can deploy applications. In accordance with the definition, it points out where the infrastructure for the deployment resides and how cloud services are made accessible to end users. This section will discuss four deployment models associated with cloud computing. [6]

3.1 Community Cloud

Community cloud computing paradigm is a promising cloud environment. This particular cloud computing model builds upon sharing computer resources from different organizations. Sharing computing resources has laid a foundation for more innovative research work. Community members can standardize operational and regulatory requirements. [7]

As stated by The National Institute of Standards and Technology (NIST), "community cloud" defines as:

“The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.” (P. Mell and T. Grance, 2011).

Benefits of using public cloud

- Cost-effective
- More control over the underlying software and hardware.
- Minimizes the likelihood of legal action due to quickly adopting regulation.

3.2 Private Cloud

A private cloud, also known as an internal or corporate cloud, refers to a cloud computing infrastructure in which all hardware and software resources are dedicated exclusively to a single customer. It is an easier option for many companies due to regulatory requirements. [9]

According to The National Institute of Standards and Technology (NIST), "private cloud" is described as:

“The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises” (P. Mell and T. Grance, 2011)

Benefits of a Private cloud:

- Complete control over hardware and software selection
- Ability to change software and hardware
- Compliance with legal requirements at anytime

3.3 Hybrid Cloud

Hybrid cloud integrates public cloud with private cloud. As a consequence, a company may execute and grow conventional or cloud-native workloads on the most appropriate computing model using a single, unified, and flexible distributed computing environment.

In accordance with The National Institute of Standards and Technology, "hybrid cloud" is described as:

"The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds)." (P. Mell and T. Grance, 2011).

Benefits of using Hybrid Cloud

- Tackle unnecessary costs.
- More efficient infrastructure.
- Greater flexibility of moving any cloud model.

3.4 Public Cloud

The public cloud is defined as computing services offered by third-party vendors. It is possible to build an entire application in the cloud or migrate from existing infrastructure. An individual or organization can use the cloud for different purposes, for instance, hosting dynamic and static websites, storing data to make a dependable backup, and using CDN for distributing content worldwide.

In accordance with The National Institute of Standards and Technology (NIST), "public cloud" defines as:

"The cloud infrastructure is provisioned for open use by the public. It may be owned, managed, and operated by a business, academic, government organization, or some combination of them. It exists on the premises of the cloud provider" (P. Mell and T. Grance, 2011).

Benefits of using the public cloud:

- No upfront investment
- Flexible capacity
- Increase Speed and agility

4 DISADVANTAGES OF CLOUD COMPUTING

Despite having such unique advantages of cloud computing, cloud computing can often be considered a complex centralized solution. In this domain, we will dive deep into a few disadvantages of cloud computing:

- **Vendor lock-in –** One of the drawbacks of cloud computing is what is known as vendor lock-in. Easy switching between cloud services has not fully matured, and enterprises may struggle to make a transition to their services from one provider to another. It can raise some unexpected costs while doing the transition.
- **Limited control-** Cloud users may discover they have less control over how services operate and are carried out within a cloud-hosted infrastructure (depending on the specific service). Since the cloud infrastructure is owned, managed, and monitored by the cloud service provider , the amount of control a customer has over their back-end infrastructure could not be the same as the cloud infrastructure.
- **Unexpected Cost-** The final downside of cloud computing is the expenses. Adopting cloud technologies on a small-scale business and for short-term projects might appear costly. Auto-scaling might add another layer of cost if it is done improperly. Allocating more resources than required definitely will result in pouring money down the drain.

4.1 How to Avoid Surprise Cost

Cloud users tend to spend a considerable amount of money on their cloud assets. Undoubtedly cloud computing cost is one of the most significant factors. Proper planning, minimizing ideal time, and use of automation help to tackle unexpected costs. However, a few methods will be described in this section to avoid incidental charges. There are several ways to prevent these unforeseen expenses while using AWS cloud services; among them, the most popular methods are serverless, notification and alert, and cost-management tools.

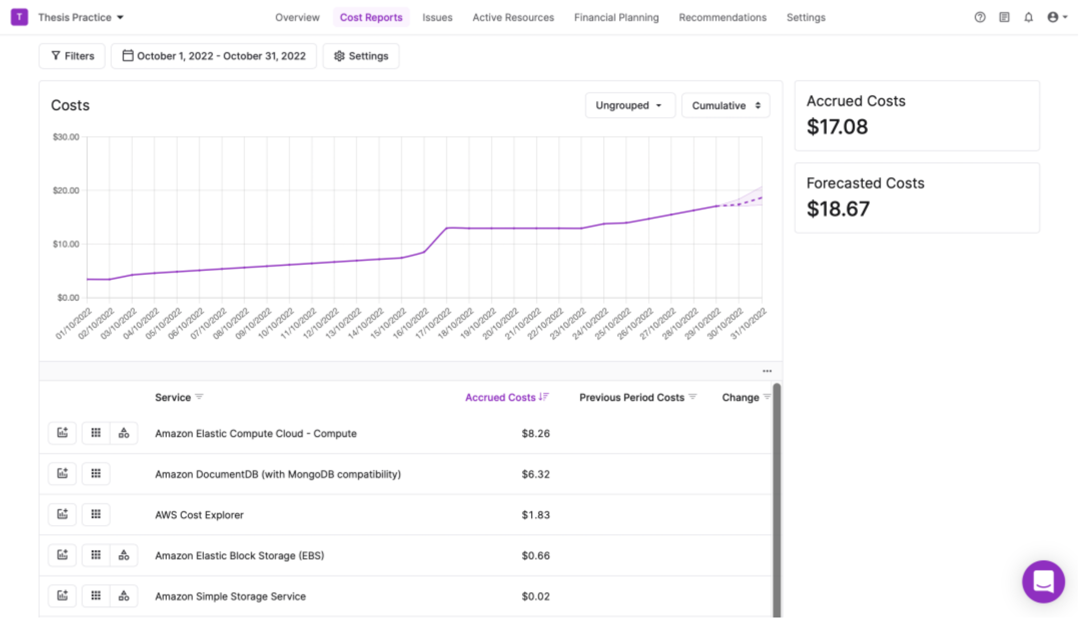
4.1.1 Third-Party Software to Visualize Cost

It is important to recognize that both AWS Billing Dashboard and third-party cost-management tools such as Vantage have been built for the same purpose. However, the two tools differ in a variety of ways. The Comparison table (TABLE 2) makes it easy to sort and quickly find the differences between the two platforms.

TABLE 2. Comparison between Vantage and AWS

	Vantage	AWS
Cost Recommendation	Vantage can optimize cost by pointing out ideal resources.	AWS does not have such a dynamic feature like Vantage.
Multi-cloud Support	Yes (All in one platform)	No (It only supports AWS services)
Resource Inventory	It can list every resource's driving costs.	Limited or no functionality of showing every driving cost.
Notification & Alert	Slack, Teams, Email	Email, SMS
Service Cost	Mostly Free	Costly

According to the comparison table, it is clearly visible that Vantage is an excellent cloud cost management and optimization platform. It supports all major cloud platforms (i.e., AWS, GCP, Azure). Since it is multicloud-supportive, cloud cost can be visualized combinedly or separately. Even though AWS has its native cost explorer(*Billing Dashboard*), Vantage is better than AWS cost explorer because of its simplicity and ease of use. Vantage has a unique UI that shows the cost right in front of the user, as shown in FIGURE 2.



Accrued Costs
$17.08

Forecasted Costs
$18.67

Service	Accrued Costs	Previous Period Costs	Change
Amazon Elastic Compute Cloud - Compute	$8.26		
Amazon DocumentDB (with MongoDB compatibility)	$6.32		
AWS Cost Explorer	$1.83		
Amazon Elastic Block Storage (EBS)	$0.66		
Amazon Simple Storage Service	$0.02		

FIGURE 2. AWS cloud Cost report visualization using Vantage

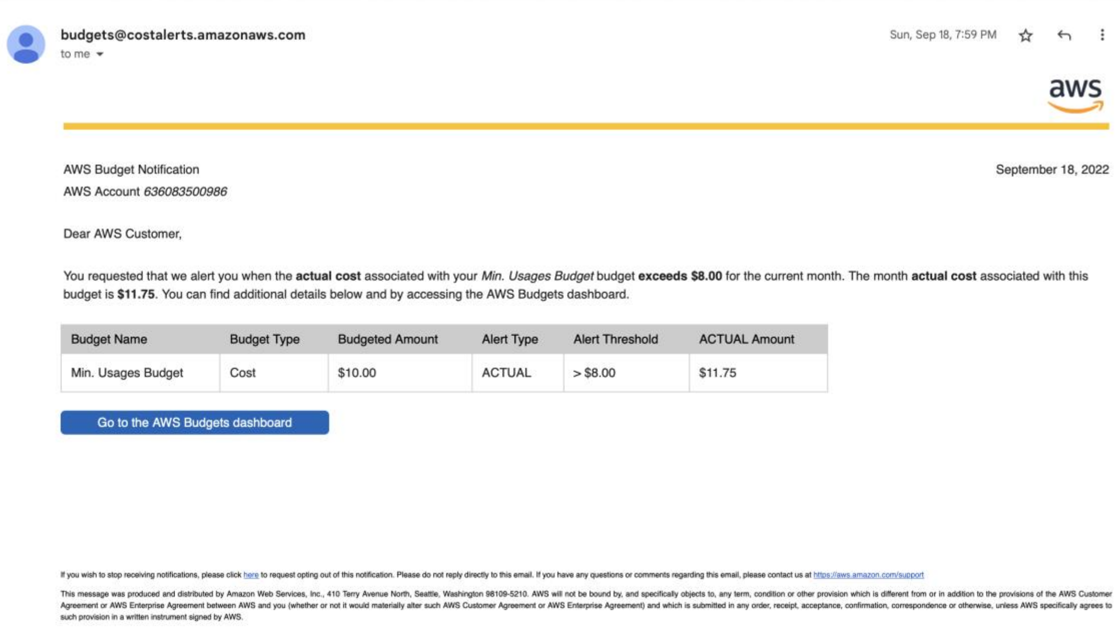
4.1.2 Setup Serverless

Serverless is a cloud-native development service that runs to respond to a particular event. In other words, it is an event-based trigger. In AWS, it has a unique name called Lambda. The Lambda function responds to demand automatically, such as starting and stopping EC2 instances at a particular time. Lambda supports numerous languages. A cloud architect [1]can setup up the system in a way that helps to shut down the system based on the input parameter. [40]

4.1.3 Budget Alert

A cloud developer can set custom budgets to track costs and usage in AWS. Based on configurations, it will send an email or text notification if it exceeds the defined threshold, as shown in the picture. An eight-dollar monthly ($8) budget has been created for demonstration purposes. After passing the threshold, it will send an email with a description. The budget alert will always break down costs, as shown in FIGURE 3.

[1] In this thesis, the word "cloud architect" is used interchangeably with "cloud developer"



budgets@costalerts.amazonaws.com
to me

Sun, Sep 18, 7:59 PM

AWS Budget Notification
AWS Account *636083500986*

September 18, 2022

Dear AWS Customer,

You requested that we alert you when the **actual cost** associated with your *Min. Usages Budget* budget **exceeds $8.00** for the current month. The month **actual cost** associated with this budget is **$11.75**. You can find additional details below and by accessing the AWS Budgets dashboard.

Budget Name	Budget Type	Budgeted Amount	Alert Type	Alert Threshold	ACTUAL Amount
Min. Usages Budget	Cost	$10.00	ACTUAL	> $8.00	$11.75

If you wish to stop receiving notifications, please click here to request opting out of this notification. Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

This message was produced and distributed by Amazon Web Services, Inc., 410 Terry Avenue North, Seattle, Washington 98109-5210. AWS will not be bound by, and specifically objects to, any term, condition or other provision which is different from or in addition to the provisions of the AWS Customer Agreement or AWS Enterprise Agreement between AWS and you (whether or not it would materially alter such AWS Customer Agreement or AWS Enterprise Agreement) and which is submitted in any order, receipt, acceptance, confirmation, correspondence or otherwise, unless AWS specifically agrees to such provision in a written instrument signed by AWS.

FIGURE 3. AWS budget alert through Email

5 OVERVIEW OF AMAZON WEB SERVICES

AWS is the most comprehensive and widely used cloud platform in the world, providing over 200 available services from various data centers across the globe. It helps to build sophisticated and reliable cloud applications with increased flexibility and scalability. Amazon EC2, Amazon Relational Database Service (RDS), Amazon Simple Storage Service (S3), Amazon CloudFront, AWS Lambda, and Amazon VPC, along with others, are the most commonly used services.
This section aims to provide an overall view of the AWS EC2, VPC, S3 & IAM because those services have been implemented in this thesis project.

5.1 Amazon EC2 (Elastic Cloud Computing)

Amazon Elastic Computation Cloud (Amazon EC2) is a web service that provides resizable computing power. It supports all major operating systems, such as Windows, Linux, and Mac. A cloud developer can scale Amazon EC2 instances horizontally or vertically based on their needs. Amazon EC2 comes up with different purchasing options based on configuration. [12]

5.2 Amazon S3 (Simple Storage Services)

Amazon S3 is a cloud storage service that allows data to be stored as objects in buckets. Architecturally S3 is designed for handling large amounts of unstructured data. Objects can be nearly any data file, such as documents, images, or videos. Buckets are logical containers for objects—theoretically, S3 stores an infinite number of objects within a bucket. According to the white paper, AWS S3 provides 11’9s durability (99.999999999 percent). [13]

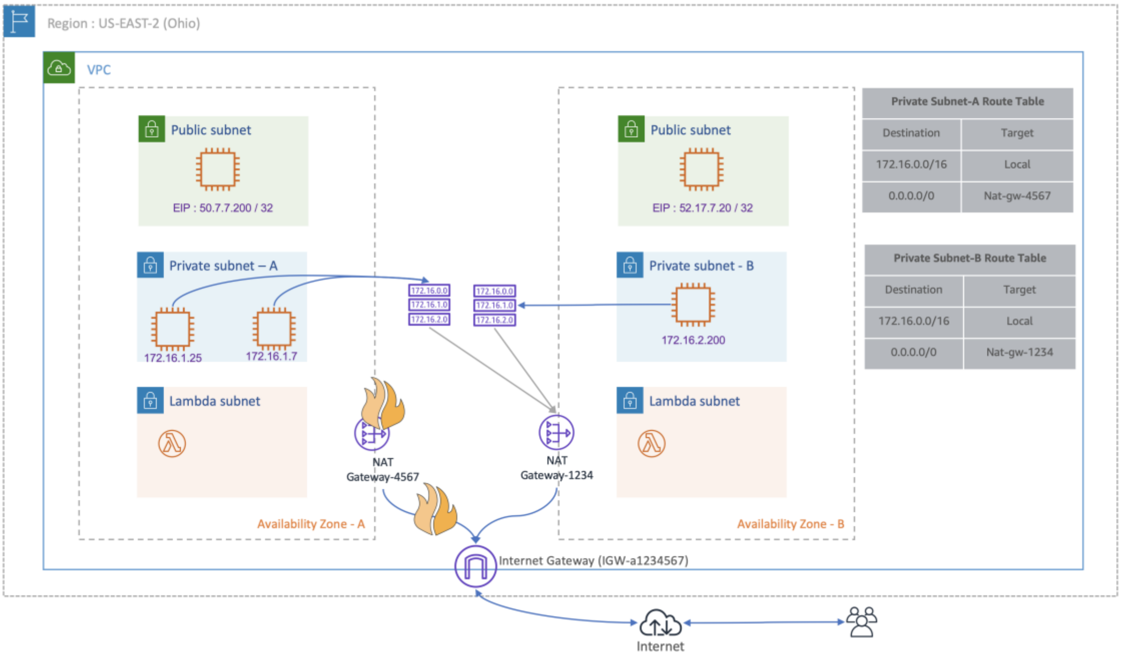
Additionally, AWS S3 offers low latency to access the data over the internet by Hypertext Transfer Protocol (HTTP) or Secure HTTP (HTTPS).

5.3 Amazon VPC (Virtual Private Cloud)

A VPC combines cloud computing resources with a VPN infrastructure, establishing a secure connection between infrastructures. Its definiens a virtual network in a logically isolated area within the AWS Cloud. FIGURE 4 shows a pair of VPCs connected to two different availability zone. A VPC can span numerous cloud servers across the globe. [23]

A VPC consists of several key components, including

- **Subnet-** In a Network topology, subnet or subnetwork refers to a network segment. Subnetting allows network traffic to move across shorter distances, which increases the effectiveness of networks. As illustrated in the figure below, a subnet's connectivity with Internet Gateway (IG) determines whether it is public or private. [16]
- **Security Group-** A Security group acts as a firewall for EC2 instances. It allows inbound or outbound traffic.



Private Subnet-A Route Table	
Destination	Target
172.16.0.0/16	Local
0.0.0.0/0	Nat-gw-4567

Private Subnet-B Route Table	
Destination	Target
172.16.0.0/16	Local
0.0.0.0/0	Nat-gw-1234

FIGURE 4. Illustration of AWS Well-Architected VPC design. [15]

- **Router-** A router is a component that routes traffic within a virtual private network (VPC).
- **Internet Gateway-** It acts as a communication channel between instances.

5.4 Identity and Access Management (IAM)

Identity and Access Management, also known as IAM, is a web service that helps to define user access. This service controls which are authenticated and authorized to use AWS services. If the access control is not properly configured, unauthorized users can swiftly abuse these same resources. Usually, the AWS root user has all the access and can be granted the same to other groups or users. [17]

6 MERN STACK APPLICATION DEVELOPMENT

MERN describes a specific set of JavaScript-based technologies used in web application development. MERN stack comprises four technologies: MongoDB, Express, React, and Node.js. It enables us to build a three-tier architecture(front-end, back-end, and database) purely with JavaScript and JSON. As a result, MERN makes the development process smoother and more manageable. [18]

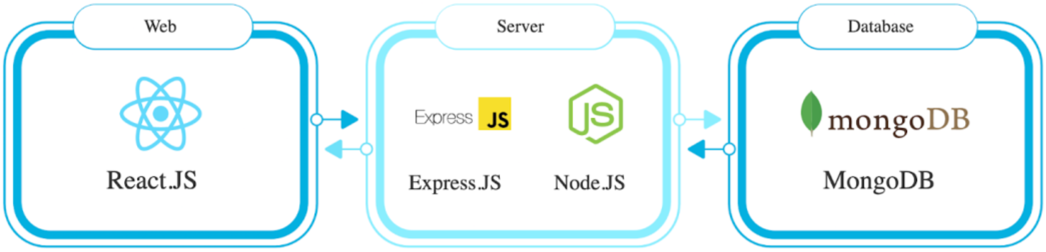


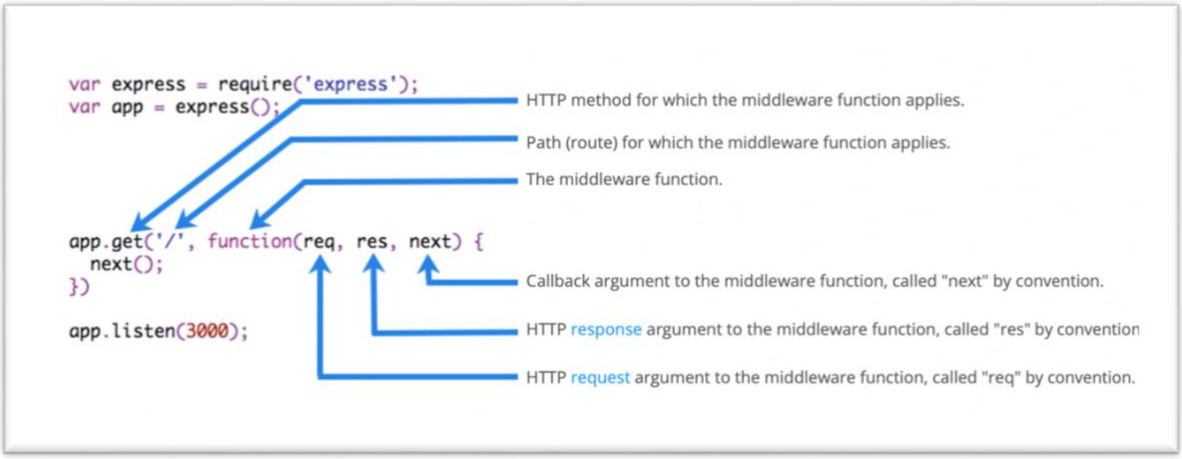
FIGURE 5. MERN Architecture. [18]

6.1 MongoDB

MongoDB is a document-oriented database. It is also a non-relational database. MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from record to record, and structure can be changed over time. MongoDB is a distributed database at its core; therefore, it has built-in high availability, horizontal scaling, and global dispersion.

6.2 Express.js

Express is a popular open-source back-end web framework developed in JavaScript and hosted on the Node.js runtime environment. As a result, Express.js can reduce coding time significantly without compromising its built quality. In 2010, Express was first introduced to the developer community and is currently on version 4.18.0. Express minimizes complexity and makes creating and maintaining apps considerably more manageable than using the built-in Node.js tools. ExpressJS accomplishes the same thing for NodeJS as Bootstrap does for HTML/CSS. It simplified NodeJS development and provided developers with new server-side functionalities. Until now, ExpressJS has been the most widely used NodeJS framework.



```
var express = require('express');
var app = express();

app.get('/', function(req, res, next) {
  next();
})

app.listen(3000);
```

HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention

HTTP request argument to the middleware function, called "req" by convention.

FIGURE 6. Express function call architecture explained. [22]

6.3 React.js

ReactJS is an open-source, component-based front-end library responsible only for the application's view layer. Facebook maintains the React library. It helps to build a user interface for any kind of web application. React is written in JavaScript. React was first released in 2013 and is one of the most commonly used JavaScript libraries among developer communities.

6.4 Node.js

Node.js is a cross-platform, open-source runtime environment for creating server-side applications. The "Single Threaded Event Loop" architecture handles multiple clients simultaneously. It is a server-side platform built on Google Chrome's JavaScript V8 Engine.Node.js application written in JavaScript and executable on Linux, macOS, and Microsoft Windows.Node.js also has an extensive library of JavaScript modules that significantly facilitates the creation of web applications using Node.js. This functionality allows developers to write synchronous and asynchronous JavaScript in a single-threaded environment without worrying about concurrency issues. To avoid complexity and handle high-volume concurrent connections, Node.js was first introduced in 2009 by Ryan Dahl. [20]

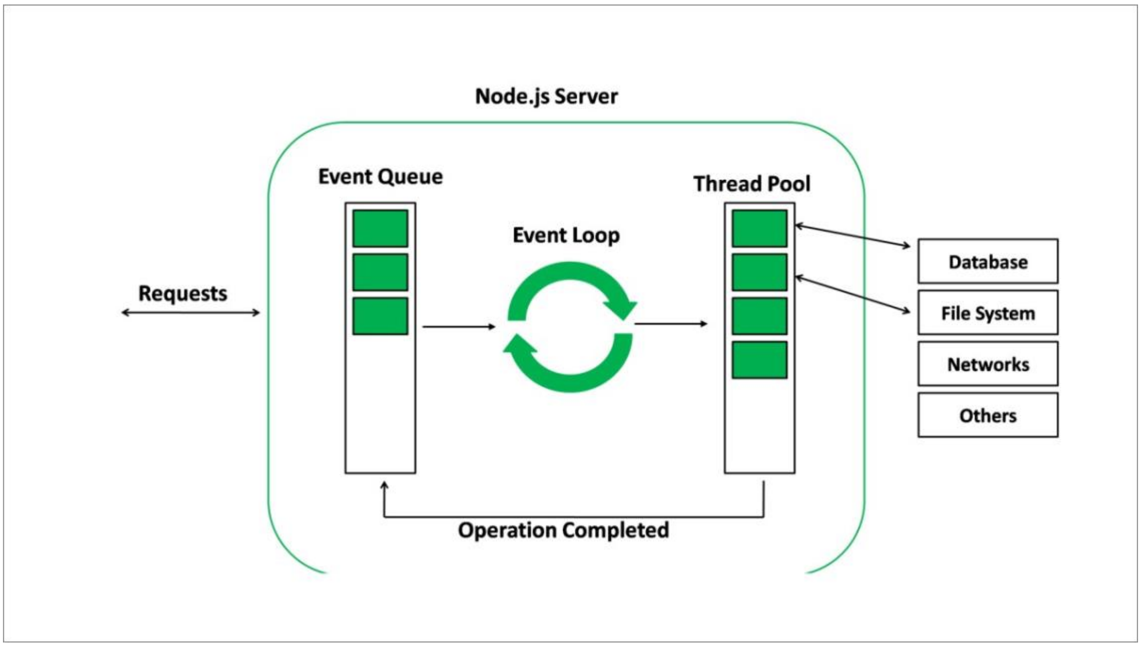
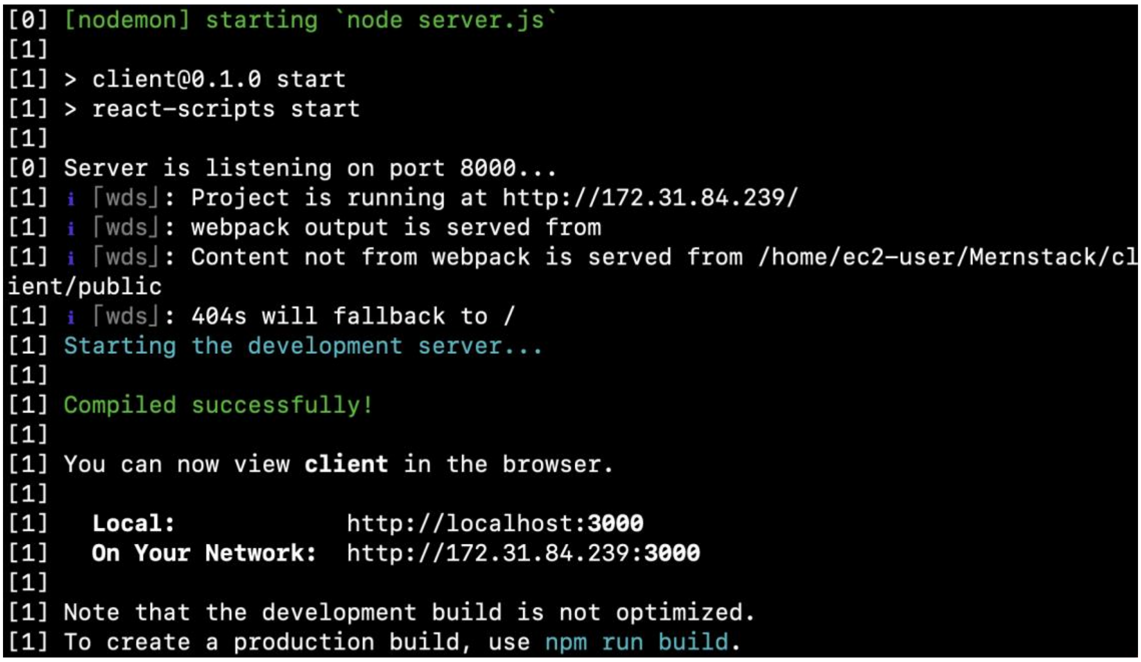


FIGURE 7. Nodejs Architecture. [20]

In this thesis demonstration MERN stack application, React has been used to build the front end, Express and Node for the back end, and Mongoose for the database. CROS functionality has been implemented to connect the backend with the front end. By entering the "npm start" command, the application will start in the development server as shown in the figure & viewable by entering the link in the browser. In this case, the URL is - http://172.31.84.238:3000



```
[0] [nodemon] starting `node server.js`
[1]
[1] > client@0.1.0 start
[1] > react-scripts start
[1]
[0] Server is listening on port 8000...
[1] ℹ ｢wds｣: Project is running at http://172.31.84.239/
[1] ℹ ｢wds｣: webpack output is served from
[1] ℹ ｢wds｣: Content not from webpack is served from /home/ec2-user/Mernstack/cl
ient/public
[1] ℹ ｢wds｣: 404s will fallback to /
[1] Starting the development server...
[1]
[1] Compiled successfully!
[1]
[1] You can now view client in the browser.
[1]
[1]   Local:            http://localhost:3000
[1]   On Your Network:  http://172.31.84.239:3000
[1]
[1] Note that the development build is not optimized.
[1] To create a production build, use npm run build.
```

FIGURE 8. MERN stack application Up & Running inside the EC2 server

7 APPROACH TO DEPLOY AN APPLICATION ONTO THE AWS CLOUD

This section describes the process of deploying full-stack applications in which a cloud architect can choose the available services that better fulfil the full-stack application architectural requirements. While designing a fault-tolerant and robust server for a large-scale production-level application, the specifications that cross a cloud developer's mind are “faster, stronger, bigger,” or possibly even “agility, speed, and scalability.” The steps below will be followed to achieve this resilient and fault-tolerant production-level server. [28]

7.1 Naming the EC2 instance

A cloud developer needs to define a suitable EC2 name for identification purposes so that an external developer in the team can identify the instance for future reference. In Amazon Web Service (AWS), this naming process is called tagging, and it is entirely optional for running an EC2 instance. By naming the instance, AWS creates *“a key-value pair*”. Where value represents the name(tag) entered for EC2. In this thesis demonstration, it is called *“Web Server,”* as illustrated in FIGURE 8. [24]

7.2 Choosing a Suitable AMI (Amazon Machine Image) & Instance Type

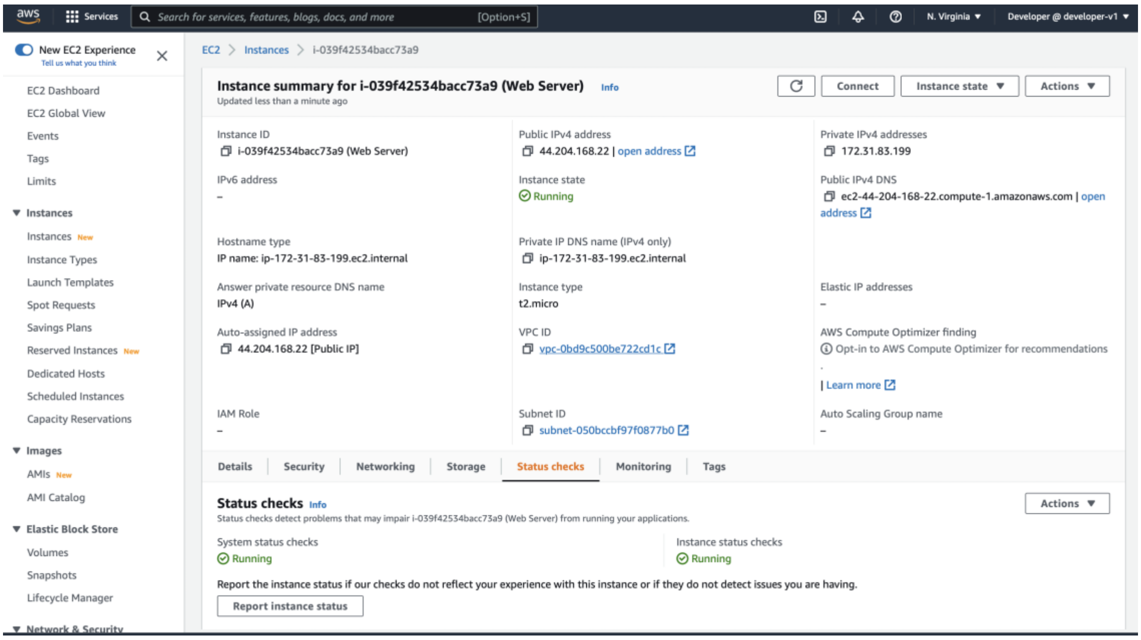
An AMI includes the operating system, application server, and software configuration needed to begin an instance. AWS has seven different kinds of OS images. Those are Ubuntu, Amazon Linux, macOS, Microsoft Windows, Red Hat, SUSE Linux, and Debian. Among them, macOS has recently been added. Amazon EC2 offers a diverse set of instance types tailored to particular use cases. Instance types are combinations of CPU, memory, storage, and networking capabilities. After careful consideration, Amazon Linux 2 AMI and t1.micro has been selected for this thesis demonstration since both have 750 hours of free usage. [29]

7.3 Key Pair Configuration

A key pair is a security credential consisting of a public and private key. It is a collection of security credentials used to authenticate while connecting to an Amazon EC2 instance. Key pair allows a cloud developer to connect instances securely using SSH(Secure Shell). After clicking the "*Create key pair*" button, it will automatically download a *.pem* format private key name "*web-server-key-pair.pem*" in the local machine, and the public key will be stored in the EC2 instance. [30]

7.4 Launching and Monitoring EC2 Instance

After configuring the EC2 instance, the developer can launch the instance by clicking the "Launch Instance" button in the user interface. Monitoring is a crucial aspect of ensuring the stability, availability, and performance of Amazon EC2 instances. The "*Status checks*" tab shown in FIGURE 9 provides a clear view of whether EC2 has identified any issues that may prevent instances from running apps.



Instance summary for i-039f42534bacc73a9 (Web Server)

Updated less than a minute ago

Instance ID: i-039f42534bacc73a9 (Web Server)

Public IPv4 address: 44.204.168.22 | open address

Private IPv4 addresses: 172.31.83.199

IPv6 address: –

Instance state: Running

Public IPv4 DNS: ec2-44-204-168-22.compute-1.amazonaws.com | open address

Hostname type: IP name: ip-172-31-83-199.ec2.internal

Private IP DNS name (IPv4 only): ip-172-31-83-199.ec2.internal

Answer private resource DNS name: IPv4 (A)

Instance type: t2.micro

Elastic IP addresses: –

Auto-assigned IP address: 44.204.168.22 [Public IP]

VPC ID: vpc-0bd9c500be722cd1c

AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. | Learn more

IAM Role: –

Subnet ID: subnet-050bccbf97f0877b0

Auto Scaling Group name: –

Status checks

Status checks detect problems that may impair i-039f42534bacc73a9 (Web Server) from running your applications.

System status checks: Running

Instance status checks: Running

Report the instance status if our checks do not reflect your experience with this instance or if they do not detect issues you are having.

FIGURE 9. The running status of the Web Server Instance

It is worth mentioning that AWS performs automated checks on every running EC2 instance to identify hardware and software issues. At the time of launching EC2 instance will automatically be associated with the default security group for the VPC.

7.5 Configuring Network Access to An Instance

To manage incoming and outgoing traffic, a security group functions as a virtual firewall for EC2 instances. Inbound rules control the incoming traffic to the instance, and outbound rules govern the outgoing traffic from an instance. When launching an instance, one or more security groups can be specified. Without specifying a security group, Amazon EC2 uses the default security group. Security group rules are modifiable. All instances connected to the security group receive automatic updates for new and changed rules.

Edit Inbound & Outbound rules

Each inbound rule consists of three key elements:

- **Protocol -** The protocol type, such as TCP or UDP. Provides an additional ICMP option.
- **Port range -** Allows traffic on a specific port or set of ports.
- **Source –** Controls the traffic that can reach the instance. It can be either a single IP address or an IP address range in CIDR notation. (For example, 10.10.1.0/28)

For this demo Full-stack application deployment, custom TCP 3000 has been used since this application is running on that PORT. Both HTTP (PORT 80) and HTTPS (PORT 443) have been enabled, as shown in TABLE 3. After allowing traffic from port 3000. Output at the EC2 instance's Public IPv4 DNS will be able to observe. In this case, the URL will be the following-

```
http://ec2-44-204-168-22.compute-1.amazonaws.com:3000
```

The outbound security group as default settings will be left as it is since this instance does not need to communicate.

TABLE 3. Inbound security group rules setup

Type	Protocol	Port Range	Source
SSH	TCP	22	0.0.0.0/0
Custom TCP	TCP	3000	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0
Custom TCP	TCP	8000	0.0.0.0/0
HTTP	TCP	80	0.0.0.0/0

7.6 NGNIX Server Setup

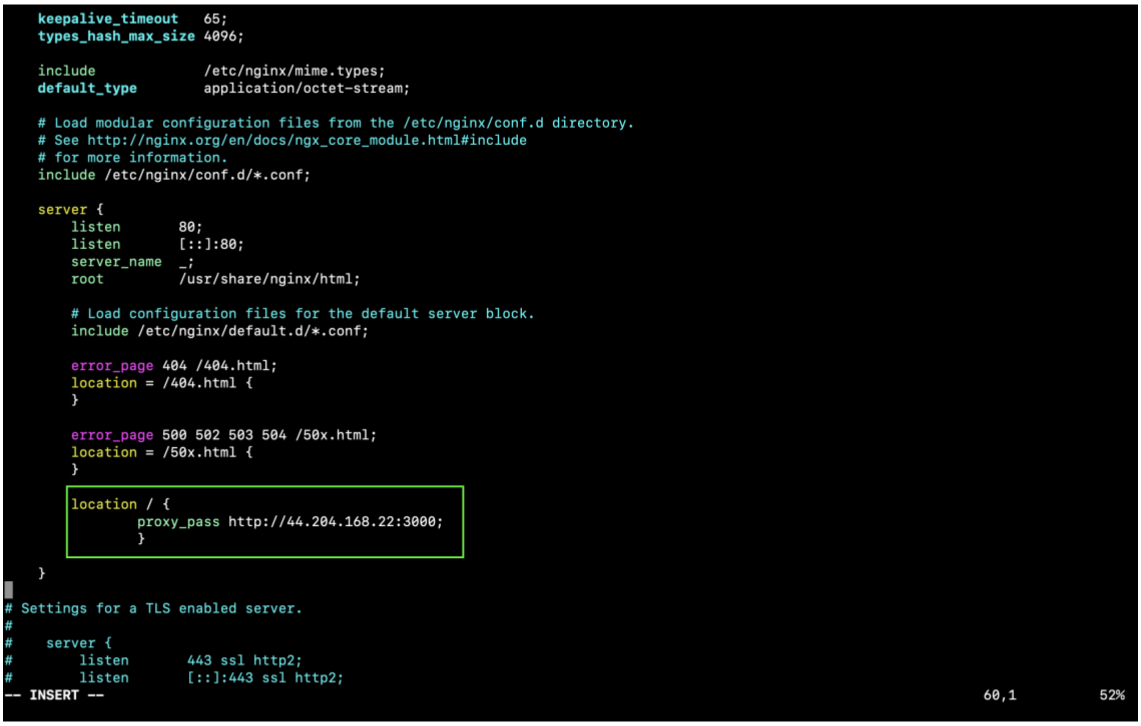
NGINX, pronounced as “*Engine X,*” is free and open-source software. NGNIX is the second most popular web server after Apache. It is best known for its optimum performance, stability, and simple configuration. NGNIX was developed by a Russian developer called *Igor Sysoev*.
It has a handful number of possible use cases, such as reverse proxying, caching, load balancing, media streaming, and many more. Along with serving as an HTTP server, NGINX may operate as an email proxy server and a load balancer. [32]
In this thesis, NGINX will serve as a reverse proxy protocol. A reverse proxy server is a protocol that often sits behind a private network's firewall and sends client requests to the proper back-end server. In other words, a reverse proxy works for the server, whereas a proxy works for the client. A reverse proxy adds another layer of abstraction and control to the network traffic flow between clients and servers. Nginx will execute on port 80 in front of the application server to intercept all internet traffic and route it to port 3000.

7.6.1 NGINX: Up & Running

In this installation guide section, a cloud developer must install open-source Nginx binaries in the Linux server by entering this “*sudo yum -y install ngnix*” command in the CLI and enabling it at the same time by using *“sudo amazon-linux extras enable nginx-1*” command. After downloading all relevant dependencies, the Nginx server will be running on Public IPv4 DNS.

Configuring NGINX

A cloud architect needs to configure Nginx to facilitate application system development and promote the achievement of functional requirements for the MERN stack application. For the best security practices, it is not recommended to have port numbers exposed in public URLs. To resolve the port number visibility issue, a proxy pass needs to add in the “*nginx.conf*” configuration file located in the */etc directory*, as shown in FIGURE 10 with the underlined green box. It is worth mentioning that the proxy pass consists of an EC2 public IPv4 address & a redirected port (*3000 in this case*)**.** After successfully implementing the code, the Nginx server needs to restart to take it into effect.

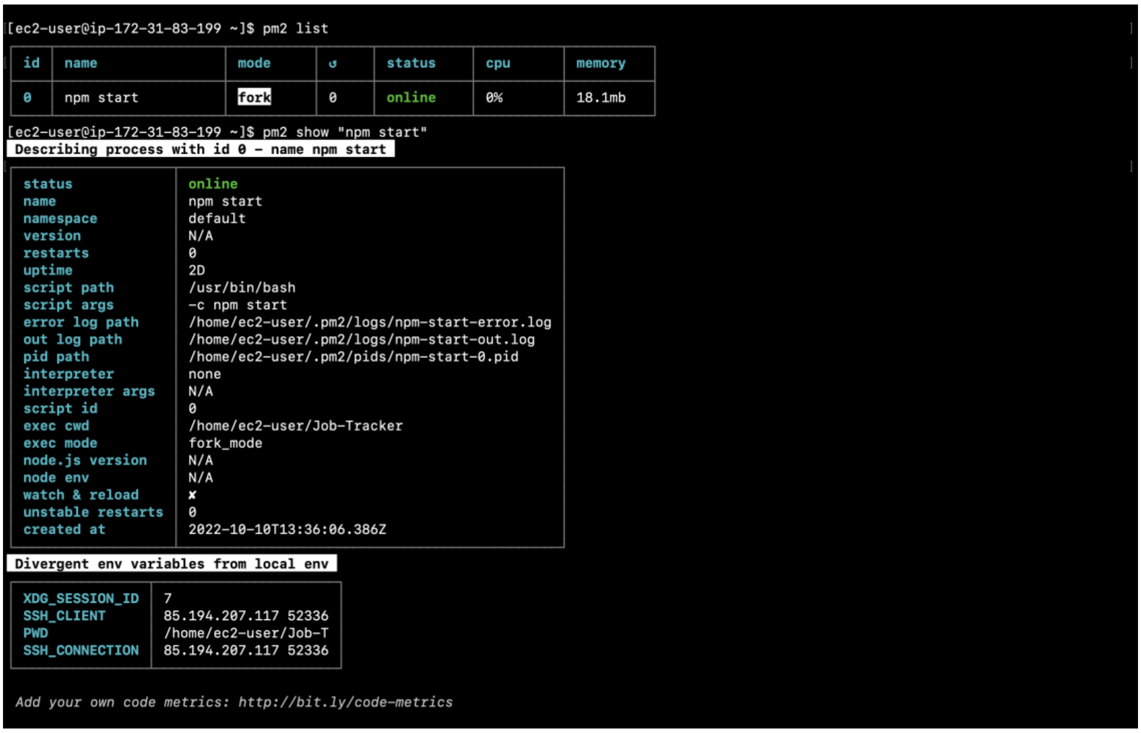


```
    keepalive_timeout   65;
    types_hash_max_size 4096;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    # Load modular configuration files from the /etc/nginx/conf.d directory.
    # See http://nginx.org/en/docs/ngx_core_module.html#include
    # for more information.
    include /etc/nginx/conf.d/*.conf;

    server {
        listen       80;
        listen       [::]:80;
        server_name  _;
        root         /usr/share/nginx/html;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        error_page 404 /404.html;
        location = /404.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }

        location / {
                proxy_pass http://44.204.168.22:3000;
                }

    }

# Settings for a TLS enabled server.
#
#    server {
#        listen       443 ssl http2;
#        listen       [::]:443 ssl http2;
-- INSERT --                                                                  60,1          52%
```

FIGURE 10. Configuration of nginx.conf file

7.7 PM2 (Process Manager)

PM2 is a Node.js process manager with an integrated load balancer. PM2 (Process manager) is a program that guarantees apps remain operational once launched. It also enables monitoring of the application. Application logs and other vital metrics, such as CPU, application status, and memory usage, can be accessed, through PM2, as shown in FIGURE 11. [34]



```
[ec2-user@ip-172-31-83-199 ~]$ pm2 list
│ id │ name        │ mode   │ ↺  │ status   │ cpu    │ memory   │
│ 0  │ npm start   │ fork   │ 0  │ online   │ 0%     │ 18.1mb   │
[ec2-user@ip-172-31-83-199 ~]$ pm2 show "npm start"
 Describing process with id 0 - name npm start 
│ status              │ online                                          │
│ name                │ npm start                                       │
│ namespace           │ default                                         │
│ version             │ N/A                                             │
│ restarts            │ 0                                               │
│ uptime              │ 2D                                              │
│ script path         │ /usr/bin/bash                                   │
│ script args         │ -c npm start                                    │
│ error log path      │ /home/ec2-user/.pm2/logs/npm-start-error.log    │
│ out log path        │ /home/ec2-user/.pm2/logs/npm-start-out.log      │
│ pid path            │ /home/ec2-user/.pm2/pids/npm-start-0.pid        │
│ interpreter         │ none                                            │
│ interpreter args    │ N/A                                             │
│ script id           │ 0                                               │
│ exec cwd            │ /home/ec2-user/Job-Tracker                      │
│ exec mode           │ fork_mode                                       │
│ node.js version     │ N/A                                             │
│ node env            │ N/A                                             │
│ watch & reload      │ ✘                                               │
│ unstable restarts   │ 0                                               │
│ created at          │ 2022-10-10T13:36:06.386Z                        │
 Divergent env variables from local env 
│ XDG_SESSION_ID │ 7                    │
│ SSH_CLIENT     │ 85.194.207.117 52336 │
│ PWD            │ /home/ec2-user/Job-T │
│ SSH_CONNECTION │ 85.194.207.117 52336 │

 Add your own code metrics: http://bit.ly/code-metrics
```

FIGURE 11. MERN Stack Application Log via PM2

7.7.1 Start Application Using PM2

One of the most important and definitive aspects of software development is deployment. A cloud developer needs to install PM2 on a Linux EC2 server as a part of a proper deployment strategy. PM2 is available through the NPM package. By entering this “*npm install pm2*” command, it will download the latest version of PM2 in the EC2 server. “*pm2 -v*” command can verify the version in the CLI. The PM2 binary (shown in FIGURE 11) is used to start an application in the background while deploying it to production. It forms a daemon that watches our application and runs it indefinitely.

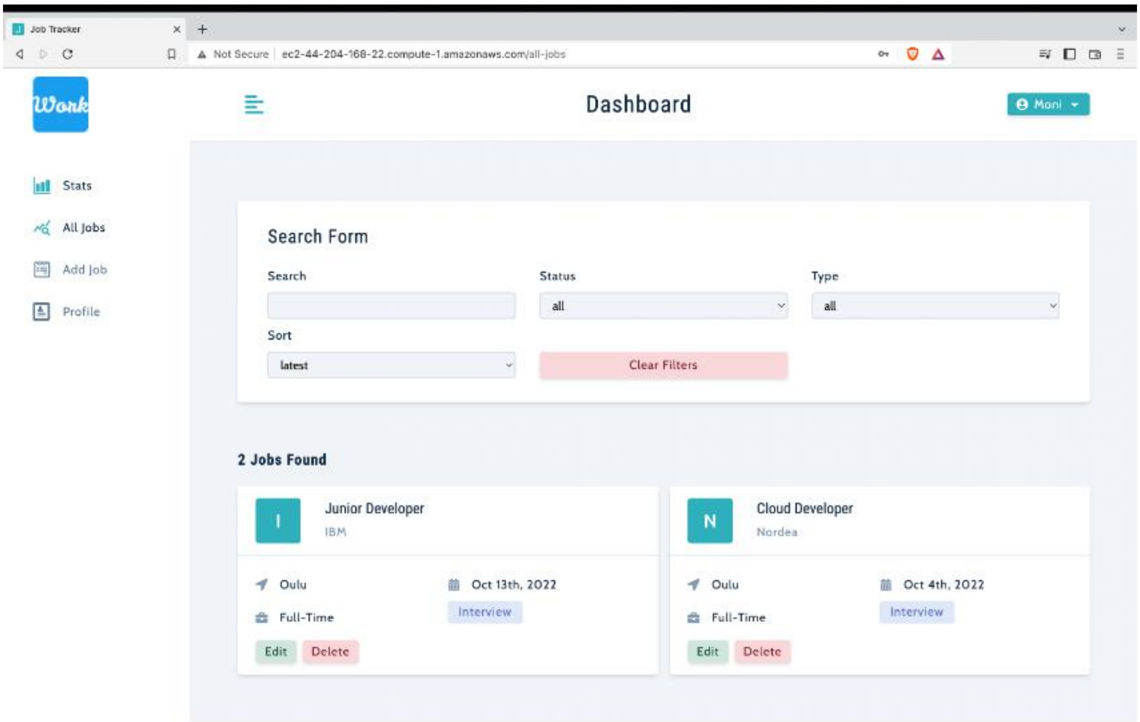
The PM2 binary (shown in FIGURE 12) is used to start an application in the background while deploying it to production. It forms a daemon that watches our application and runs it indefinitely.

```
pm2 start “npm start” --name "job Tracker"
```

FIGURE 12. PM2 command to run the MERN stack application

7.8 Results of The Deployment

Upon completing a successful connection with MongoDB, implement the Nginx proxy protocol. MERN stack application will be successfully functional, as shown in FIGURE 13, and PM2 will help to keep the system up and running infinitely.



ec2-44-204-168-22.compute-1.amazonaws.com/all-jobs

Search Form

Search:

Status: all

Type: all

Sort: latest

2 Jobs Found

I Junior Developer
IBM
Oulu
Oct 13th, 2022
Full-Time
Interview

N Cloud Developer
Nordea
Oulu
Oct 4th, 2022
Full-Time
Interview

FIGURE 13. Deployed application in AWS Linux EC2

8 INFRASTRUCTURE AS A CODE

Infrastructure as code, also known as IaC, offers DevOps practices that enable a cloud developer to work more efficiently with configuration files rather than graphical UI. IaC allows a cloud developer to declare resource configurations that may subsequently be versioned, reused, and shared. The most widely used configuration management and provisioning tools are as follows: Terraform, Ansible, CloudFormation, Chef, and Puppet.

TABLE 4. A comparison of IaC as of October 2022. [36]

Name	Release year	Source
Terraform	2014	Open
Ansible	2012	Open
CloudFormation	2011	Closed
Chef	2009	Open
Puppet	2005	Open

Terraform is an open-source IaC tool created and maintained by HashiCrop. It uses declarative configuration language known as HashiCorp Configuration Language (HCL) and Terraforms language code is stored with the "*.tf*" file extension.

8.1 Why Use Terraform Over Others?

Infrastructure as code (IaC) is fantastic, but the process of picking an IaC tool is not. Many of the IaC tools overlap in what they do. Perhaps they have so little difference that using one of them does not make a significant difference. Yet the most important reason to choose Terraform over others is as follows-

- **Multicloud Deployment-** One of the main reasons for choosing Terraform is its multi-cloud support. With Terraform, a developer can handle cross-cloud dependencies and manage various providers using the same procedure. For massive, multi-cloud systems, this makes administration and orchestration simpler. Terraform support major cloud provider, including Amazon AWS, Google GCP, and Microsoft Azure.

- **Declarative Language-** Terraform is declarative in nature, where a developer can specify the end state. The IaC tool is in charge of determining how to achieve that condition.

```
-ec2:
  count       = 20
  image       = "ami-09d3b3274b6c5d4aa"
  instance_type  = "t3.micro"
```

FIGURE 14. Ansible template to deploy EC2 onto AWS

```
resource "aws_instance" "Oulu10_server" {
  count       = 20
  ami         = "ami-09d3b3274b6c5d4aa"
  instance_type = "t3.micro"
}
```

FIGURE 15. Terraform template to deploy EC2 onto AWS

On the surface, these two approaches will produce similar results. The exciting thing will happen if a cloud developer wants to make any changes due to any possible scenarios (i.e., increasing traffic). With Ansible (FIGURE 14), if a cloud developer runs this code with an increasing number, let's say the total *count*=*30*. It will add 30 servers on top of 20 servers. So, in total, 50 servers will be deployed.

On the other hand, in Terraform (FIGURE 15), *count*=*30* will be considered as the final counting. It will automatically apply the configuration and update it to 30 servers.

- **Agentless-** Agent is a little piece of software that can monitor IT infrastructure. Both Agentless and Agent-Based monitoring methods have been around for decades. Agent-Based monitoring requires us to install small pieces of software on the equipment. The agent will collect all the Metrics locally and send them to the monitoring platform. In contrast, Agentless solutions require that the equipment has some kind of API or protocol for monitoring platforms to collect the metrics remotely; therefore, no need to install an agent beforehand. DevOps teams are monitoring one hundred percent of the infrastructure, not just servers but databases, networks, firewalls, middleware, wi-fi, and everything. They need universal equipment that can work with all equipment without having to mess-around. As Terraform users, we do not have to take care of that Agent software. Cloud providers like Amazon AWS, Google GCP, and Microsoft AZURE will manage agent software on each of their physical servers.

8.2 Practical Implementation of Terraform Using Amazon AWS

Terraform is a powerful tool in automating deployment across multiple clouds. In this domain, a step-by-step process will be described to build, change, and destroy AWS infrastructure using Terraform.

8.2.1 Main Terraform command

Terraform CLI to Terraform is the *"terraform"* command, which accepts a wide range of subcommands. In terms of practical implementation and in terms of usages of terraform effectively, five commands are described below without any additional arguments.

```
“Terraform init”- command will initialize a working directory.

“Terraform validate”- will validates the terraform file syntax.

“Terraform plan”- will create a reviewable execution plan.

“Terraform apply”- command will execute the action plan.

“Terraform destroy”- command will destroy all previously created remote objects.
```

8.2.2 Default Profile Setup For AWS

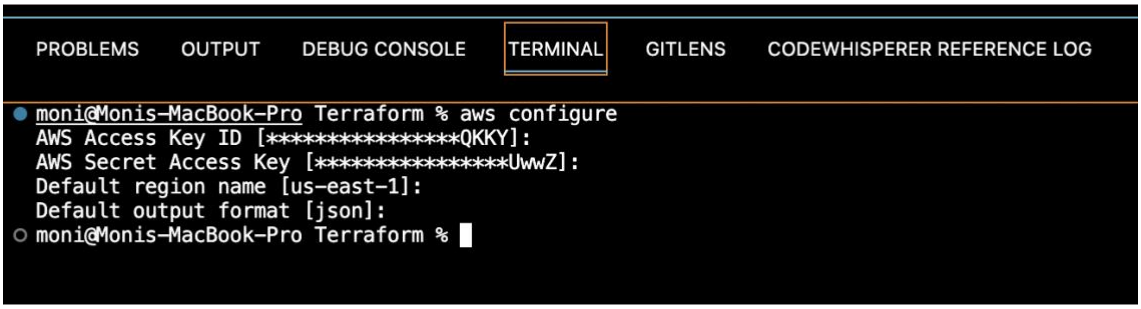
After installing terraform on a local machine, a cloud developer can confirm this by running the following command *"terraform -v"* in the terminal. Upon successfully installed, this command will print the current version of the terraform.

It is a must to provide the Amazon AWS access key and secret key in Terraform configuration file so that it can interact with the cloud account in order to deploy, modify and destroy servers. There are two ways to provide those keys (access, secret), one in the *main.tf* file and another in the configuration file. By providing keys in the main file, it imposes two most significant risks factors which are-

One- In case of deletion, a cloud developer needs to reproduce the key & copy-paste each time while interacting with AWS cloud.

Two- Another reason is if it is somehow compromised, anyone can do anything with the AWS cloud account. This is key simply too powerful.

In order to avoid those risks, it is recommended to make a default profile by using the command *"vim ~/.aws/credentials"*. After storing the credential, the *"aws configure"* command will print the following in the console, as demonstrated in FIGURE 16 as a confirmation. After configuring the default profile, we can use **profile = "default"** as a reference for our access & secret key.



```
● moni@Monis-MacBook-Pro Terraform % aws configure
  AWS Access Key ID [****************QKKY]:
  AWS Secret Access Key [****************UwwZ]:
  Default region name [us-east-1]:
  Default output format [json]:
○ moni@Monis-MacBook-Pro Terraform %
```

FIGURE 16. AWS default configuration verification for Terraform

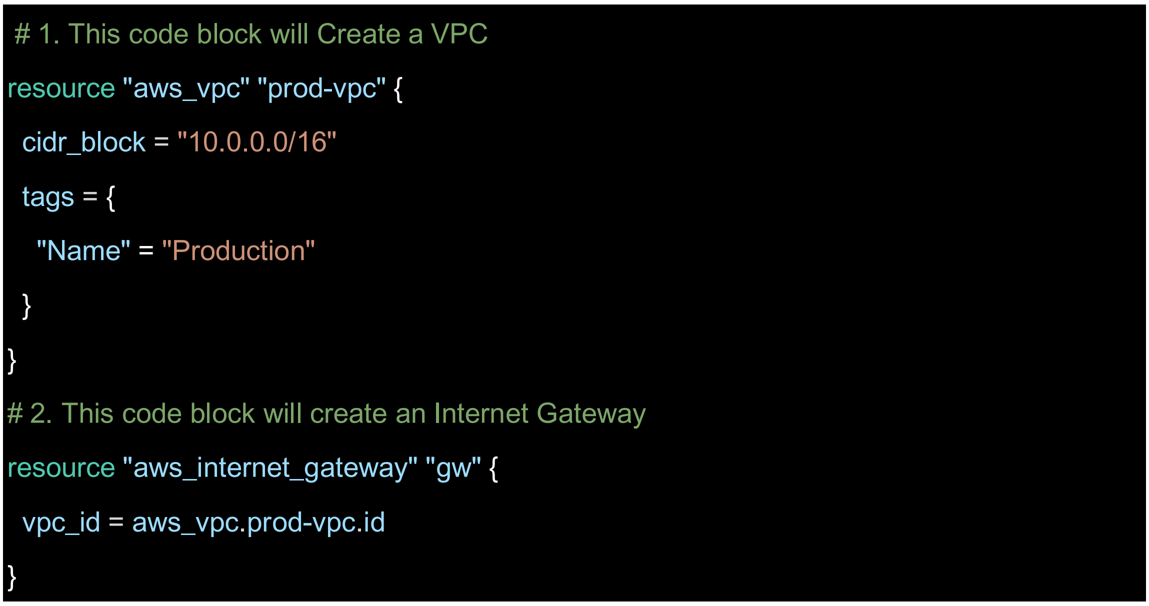
8.2.3 Defining a Provider

The first thing a developer has to do is to define a provider (AWS, in this thesis project case). It is a plugin that allows Terraform to talk specific set of APIs. It will make sure to download all necessary dependencies to talk to the AWS API.

8.2.4 Create a Virtual Private Cloud (VPC) & Internet Gateway (IG)

The following block of code will be used to create a VPC. In order to do this, a cloud developer needs to specify a CIDR block that will define the range of IP addresses. In this implementation, 10.0.0.0/16 will create 65,536 addresses theoretically. It is worth mentioning that AWS has five reserved IP addresses, so that it will make 65,531 addresses instead of 65,536.

For the resources in a VPC to send and receive traffic from the internet, an internet gateway must be attached to the VPC. An internet gateway enables resources to connect to the internet. The following code block, as shown in FIGURE 17, will create and attach an IG with the VPC called "*Production.*"

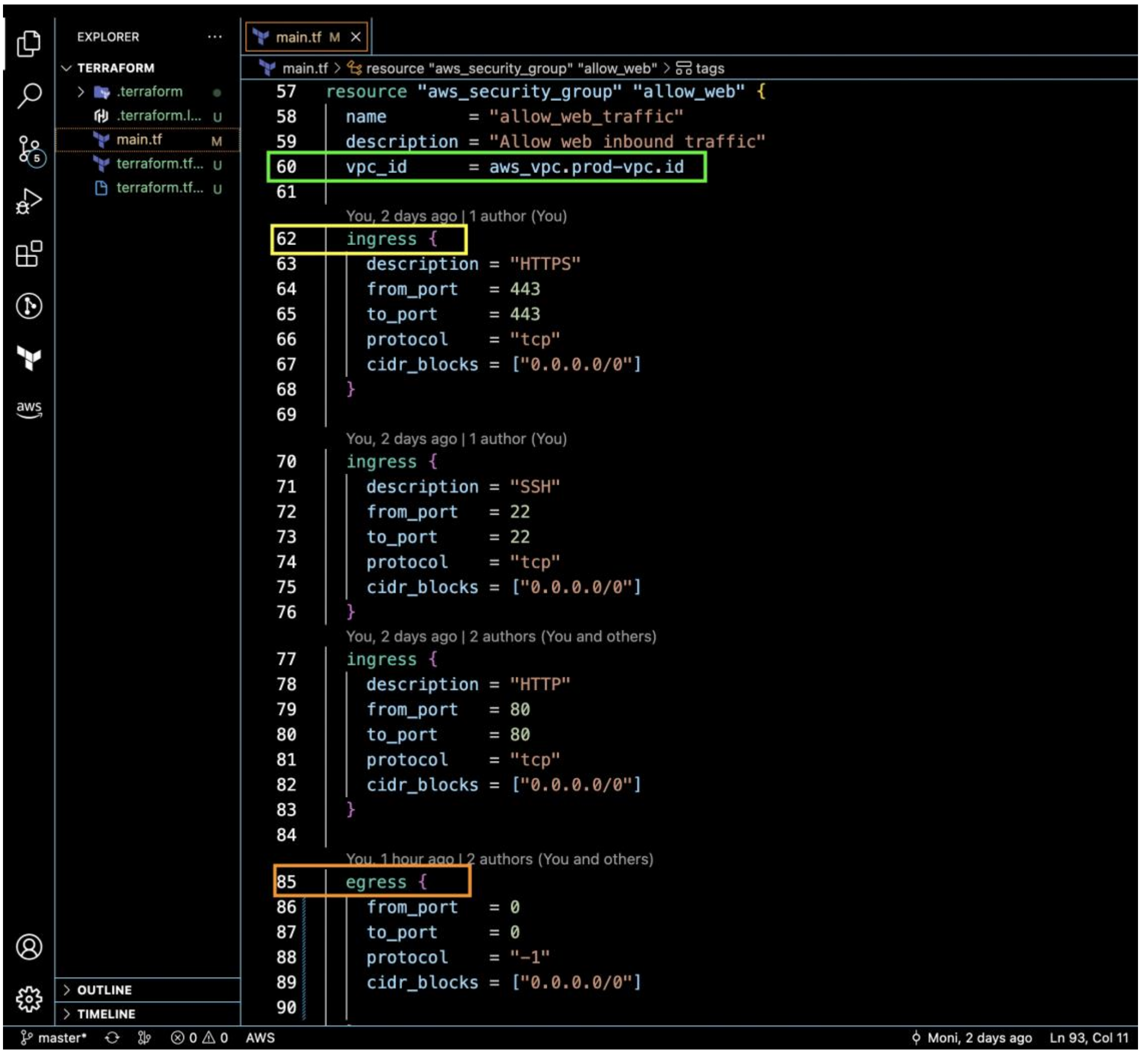


```
 # 1. This code block will Create a VPC
resource "aws_vpc" "prod-vpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    "Name" = "Production"
  }
}
# 2. This code block will create an Internet Gateway
resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.prod-vpc.id
}
```

FIGURE 17. VPC and IG creation process in Terraform

8.2.5 Security Group Creation

A security group is a way to define the network traffic for the EC2 instances. Terraform currently provides both a standalone *Security Group* rule resource and a *Security Group* resource with in-line defined ingress and egress rules. A name is a must for creating a security group so that it can be used as a future reference. The security group's name in this practical section is *"allow_web"* as shown in FIGURE 18. Security Group name cannot be edited after the resource is created.
In this creation process, the VPC reference is attached to the security group with the highlighted syntax, as shown in line 60. Ingress stands for inbound rules. For the security group, three ports have been added to the rules, for example, 443 for HTTPS, 80 for HTTP, and PORT 22 for SSH. Defining *"0.0.0.0/0"* in the CIDR block means any IP address can access it. For the outbound rules, as described in line 85 with the highlighted orange box in FIGURE 18. "*from_port = 0*" and "*to_port=0*" means allowing all ports in the egress direction. "*Protocol= -1*" is used to specify all protocols.



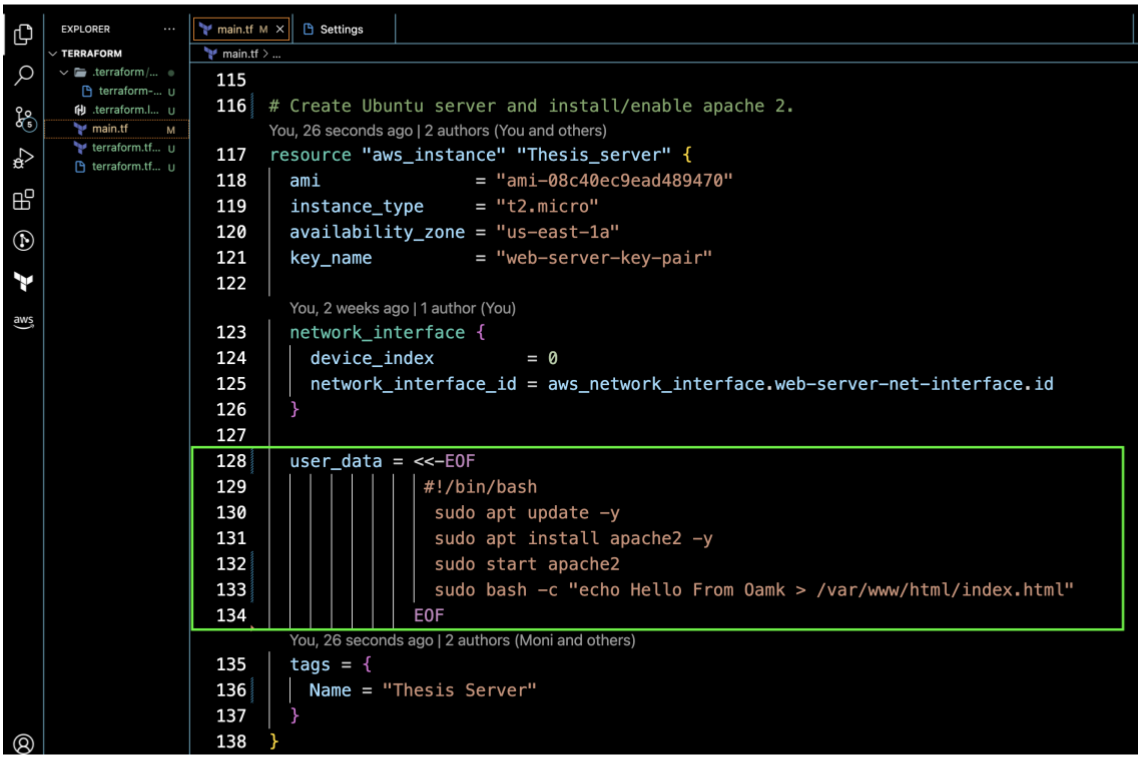
```
resource "aws_security_group" "allow_web" {
  name        = "allow_web_traffic"
  description = "Allow web inbound traffic"
  vpc_id      = aws_vpc.prod-vpc.id

  You, 2 days ago | 1 author (You)
  ingress {
    description = "HTTPS"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  You, 2 days ago | 1 author (You)
  ingress {
    description = "SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  You, 2 days ago | 2 authors (You and others)
  ingress {
    description = "HTTP"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  You, 1 hour ago | 2 authors (You and others)
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
```

FIGURE 18. Security group construction using Terraform

8.2.6 Create an Ubuntu Server and Install Apache

Ubuntu is a Linux-based operating system, and *Apache* is an open-source web server available for Linux servers. In this creation process, we have to define Ami, instance type, key name, and an availability zone. Key name and availability zone are optional, but we need to define the key name in order to SSH. The hardcoded availability zone will make sure all services are created in the same region.

Highlighted user_data code syntax starting from line 128, as shown in FIGURE 19 will install and activate the *Apache* server. “Hello from Oamk” will be delivered upon visiting the IP address. (FIGURE 22)

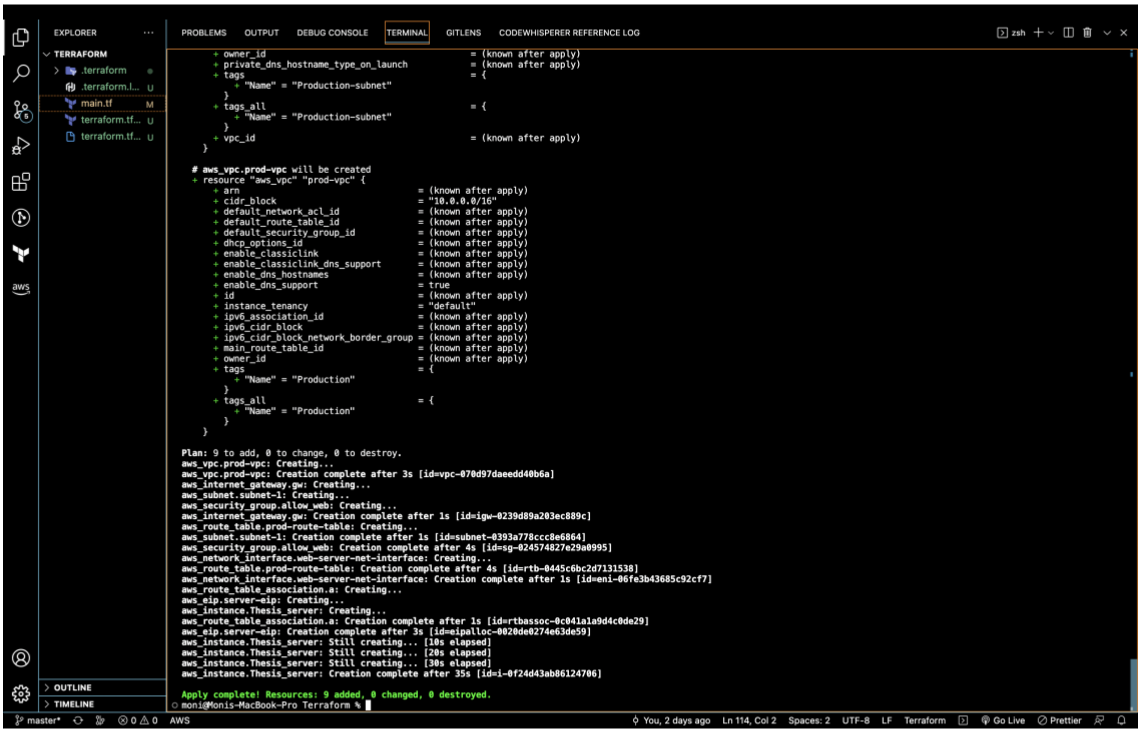


```

# Create Ubuntu server and install/enable apache 2.
You, 26 seconds ago | 2 authors (You and others)
resource "aws_instance" "Thesis_server" {
  ami               = "ami-08c40ec9ead489470"
  instance_type     = "t2.micro"
  availability_zone = "us-east-1a"
  key_name          = "web-server-key-pair"

  You, 2 weeks ago | 1 author (You)
  network_interface {
    device_index         = 0
    network_interface_id = aws_network_interface.web-server-net-interface.id
  }

  user_data = <<-EOF
              #!/bin/bash
               sudo apt update -y
               sudo apt install apache2 -y
               sudo start apache2
               sudo bash -c "echo Hello From Oamk > /var/www/html/index.html"
              EOF
  You, 26 seconds ago | 2 authors (Moni and others)
  tags = {
    Name = "Thesis Server"
  }
}
```

FIGURE 19. Apace server installation process during boot event

8.3 Result of The Deployment Using Terraform

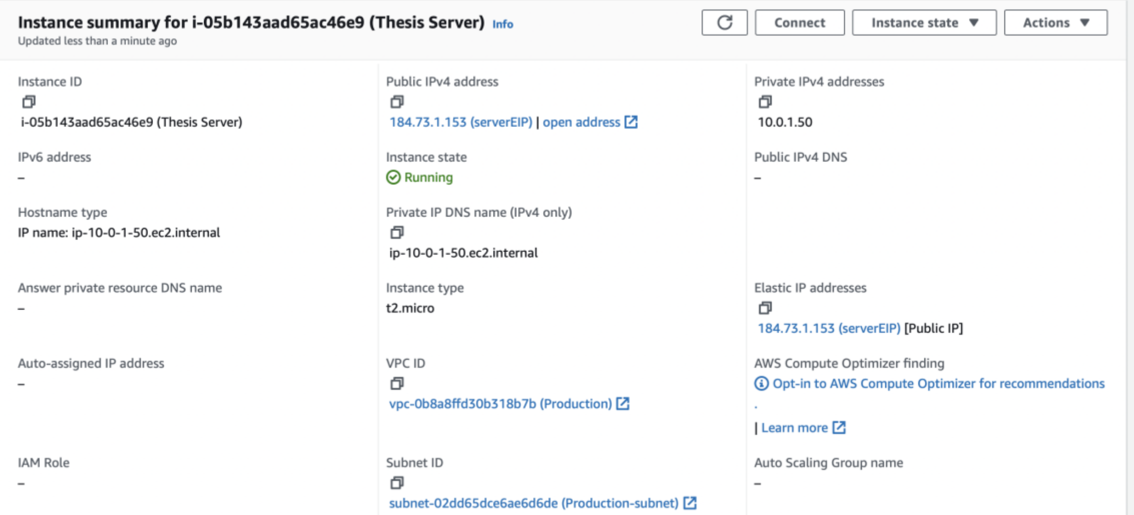
First, the “*terraform init”* command has to be run. Terraform will download all relevant dependencies and initialize a working directory containing the configuration file. The “*Validate*” command needs to be executed before the final approach. The “*validate*” command is always recommended, even though the “*terraform apply”* command will always validate the whole process before execution. After validating, we can finally run terraform apply -auto-approve . *“-auto-approve”* subcommand will help to avoid unnecessary typing in the console. Upon successfully running, it will create all the defined services in the code block & confirmation message will be displayed in bold green text.



```
      + owner_id                                   = (known after apply)
      + private_dns_hostname_type_on_launch        = (known after apply)
      + tags                                       = {
          + "Name" = "Production-subnet"
        }
      + tags_all                                   = {
          + "Name" = "Production-subnet"
        }
      + vpc_id                                     = (known after apply)
    }

  # aws_vpc.prod-vpc will be created
  + resource "aws_vpc" "prod-vpc" {
      + arn                                  = (known after apply)
      + cidr_block                           = "10.0.0.0/16"
      + 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_classiclink                   = (known after apply)
      + enable_classiclink_dns_support       = (known after apply)
      + enable_dns_hostnames                 = (known after apply)
      + enable_dns_support                   = true
      + 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" = "Production"
        }
      + tags_all                             = {
          + "Name" = "Production"
        }
    }

Plan: 9 to add, 0 to change, 0 to destroy.
aws_vpc.prod-vpc: Creating...
aws_vpc.prod-vpc: Creation complete after 3s [id=vpc-070d97daeedd40b6a]
aws_internet_gateway.gw: Creating...
aws_subnet.subnet-1: Creating...
aws_security_group.allow_web: Creating...
aws_internet_gateway.gw: Creation complete after 1s [id=igw-0239d89a203ec889c]
aws_route_table.prod-route-table: Creating...
aws_subnet.subnet-1: Creation complete after 1s [id=subnet-0393a778ccc8e6864]
aws_security_group.allow_web: Creation complete after 4s [id=sg-024574827e29a0995]
aws_network_interface.web-server-net-interface: Creating...
aws_route_table.prod-route-table: Creation complete after 4s [id=rtb-0445c6bc2d7131538]
aws_network_interface.web-server-net-interface: Creation complete after 1s [id=eni-06fe3b43685c92cf7]
aws_route_table_association.a: Creating...
aws_eip.server-eip: Creating...
aws_instance.Thesis_server: Creating...
aws_route_table_association.a: Creation complete after 1s [id=rtbassoc-0c041a1a9d4c0de29]
aws_eip.server-eip: Creation complete after 3s [id=eipalloc-0020de0274e63de59]
aws_instance.Thesis_server: Still creating... [10s elapsed]
aws_instance.Thesis_server: Still creating... [20s elapsed]
aws_instance.Thesis_server: Still creating... [30s elapsed]
aws_instance.Thesis_server: Creation complete after 35s [id=i-0f24d43ab86124706]

Apply complete! Resources: 9 added, 0 changed, 0 destroyed.
○ moni@Monis-MacBook-Pro Terraform %
```

FIGURE 20. Results of Terraform execution

According to the practical approach, the following will be shown, an EC2 instance running in the Amazon AWS called "Thesis Server" (FIGURE 21). "Hello from Oamk" will be delivered upon visiting the IP address (FIGURE 22). Ping command will be able to run after logging from the console (FIGURE 23). On top of it, the route table, CIDR block, IAM, VPC, and IG will also be created in the Amazon AWS cloud, and all will be attached to the "Thesis Server" to make it functional.



Instance summary for i-05b143aad65ac46e9 (Thesis Server)
Updated less than a minute ago

Instance ID
i-05b143aad65ac46e9 (Thesis Server)

IPv6 address
–

Hostname type
IP name: ip-10-0-1-50.ec2.internal

Answer private resource DNS name
–

Auto-assigned IP address
–

IAM Role
–

Public IPv4 address
184.73.1.153 (serverEIP) | open address

Instance state
Running

Private IP DNS name (IPv4 only)
ip-10-0-1-50.ec2.internal

Instance type
t2.micro

VPC ID
vpc-0b8a8ffd30b318b7b (Production)

Subnet ID
subnet-02dd65dce6ae6d6de (Production-subnet)

Private IPv4 addresses
10.0.1.50

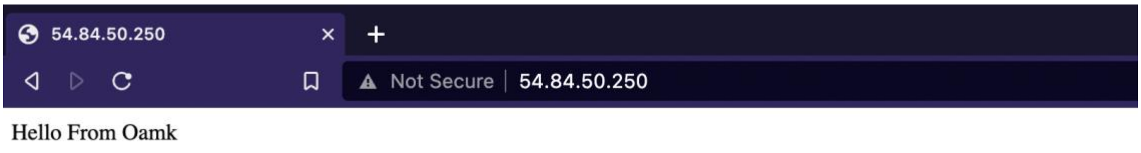
Public IPv4 DNS
–

Elastic IP addresses
184.73.1.153 (serverEIP) [Public IP]

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations
. | Learn more

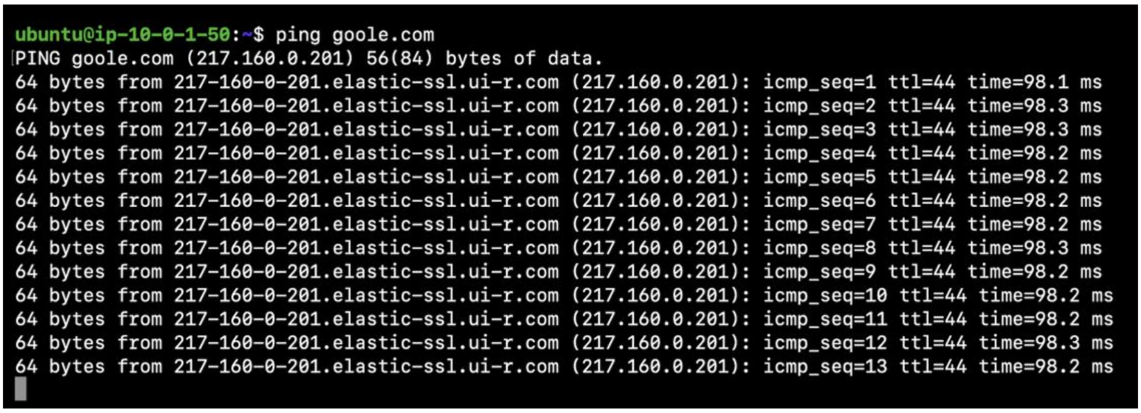
Auto Scaling Group name
–

FIGURE 21. Newly deployed EC2 Instance summary



Hello From Oamk

FIGURE 22. HTML text in the deployed “thesis server” using Terraform



```
ubuntu@ip-10-0-1-50:~$ ping goole.com
PING goole.com (217.160.0.201) 56(84) bytes of data.
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=1 ttl=44 time=98.1 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=2 ttl=44 time=98.3 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=3 ttl=44 time=98.3 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=4 ttl=44 time=98.2 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=5 ttl=44 time=98.2 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=6 ttl=44 time=98.2 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=7 ttl=44 time=98.2 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=8 ttl=44 time=98.3 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=9 ttl=44 time=98.2 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=10 ttl=44 time=98.2 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=11 ttl=44 time=98.2 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=12 ttl=44 time=98.3 ms
64 bytes from 217-160-0-201.elastic-ssl.ui-r.com (217.160.0.201): icmp_seq=13 ttl=44 time=98.2 ms
```

FIGURE 23. Ping the command log of the ubuntu server

9 CONCLUSION AND FUTURE DEVELOPMENT

Cloud-native technology is an emerging solution that promises high scalability of applications and software. The aim of this thesis was to study AWS services and implement them to deploy MERN stack applications and test them in the cloud. Instead of using a manual deployment process, Terraform provides a complete solution to configure and build local and remote instances and saves a lot of time while developing the application.

Despite the promising results in deploying the MERN stack application in the cloud, only the Amazon AWS cloud environment was tested. It could have been an excellent idea if another cloud provider like Google GCP and Microsoft Azure was included to conduct the application deployment and test their performance simultaneously. Stopped or hibernated instances always receive a new public IP address; this problem resulted in a significant loss of time that might have been spent to enhance the implementation's features. If something had been changed during this project, it would have been to spend less time configuring the NGINX on the remote server.

The project covers a tiny fraction of what AWS and Terraform can do together. Much more can be done by leveraging this technology. One of the most important use cases that will be developed in the future is to set up a CI/CD pipeline with GitHub to automate the workflow so that when the project is updated, it can fetch the latest version from GitHub. Organizations can produce higher-quality code more quickly by automating CI/CD throughout the development, testing, production, and monitoring phases of the software development lifecycle.

After working with AWS services, it became apparent that it was not designed for cost optimization for the short-term use case. Nevertheless, it was possible to achieve significant cost deduction just by reserving the services for a long-term period. It would be an excellent idea if such a move were made. Although, it is possible to execute each of the deployment steps manually. However, the actual value of Terraform is realized through automation. Automating the process aims to minimize human error and maintain a consistent approach to releasing software.

REFERENCES

1. The NIST Definition of Cloud Computing. Data retrieved on 3rd October 2022 from https://csrc.nist.gov/publications/detail/sp/800-145/final

2. Types of Cloud Computing. Data retrieved on 4th October 2022 from https://aws.amazon.com/types-of-cloud-computing/

3. AWS Market Share 2022(Q2). Data retrieved on 4th October 2022 from https://www.wpoven.com/blog/aws-market-share/

4. A systematic mapping study of infrastructure as code research. Data retrieved on 4th October 2022 from https://www.sciencedirect.com/science/article/abs/pii/S0950584918302507

5. What is IaaS? Infrastructure as a Service Explained. Data retrieved on 3rd October 2022 from https://www.digitalocean.com/blog/what-is-iaas

6. Cloud Deployment Models. Data retrieved on 5th November 2022 from https://www.sciencedirect.com/topics/computer-science/cloud-deployment-model

7. What is Community Cloud. . Data retrieved on 5th November 2022 from https://www.spiceworks.com/tech/cloud/articles/what-is-community-cloud/

8. PaaS(Platform as a service). Data retrieved on 4th October 2022 from https://www.ibm.com/cloud/learn/paas

9. Private Cloud. Data retrieved on 31st October 2022 from https://www.ibm.com/cloud/learn/introduction-to-private-cloud

10. Disadvantage of Cloud computing. Data retrieved on 31st October 2022 from https://cloudacademy.com/blog/disadvantages-of-cloud-computing/

11. IBM Cloud,2021:IaaS versus PaaS versus SaaS .Data retrieved on 21st October 2022 from https://www.ibm.com/cloud/learn/iaas-paas-saas#toc-saas-versu-8QqZtSK8

12. What is EC2? Data retrieved on 8th November 2022 from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

13. Amazon S3 Features. Data retrieved on 5th November 2022 from.https://aws.amazon.com/s3/features/

14. Central storage: Amazon S3 as the data lake storage platform.Data retrieved on 3rd November 2022 from https://docs.aws.amazon.com/pdfs/whitepapers/latest/building-data-lakes/building-data-lakes.pdf#amazon-s3-data-lake-storage-platform

15. AWS Architecture Blog (One to Many: Evolving VPC Design). Data retrieved on 3rd November 2022 from https://aws.amazon.com/blogs/architecture/one-to-many-evolving-

vpc-design/

16. VPC with public and private Subnets(NAT) .Data retrieved on 3rd November 2022 from https://docs.aws.amazon.com/pdfs/vpc/latest/userguide/vpc-ug.pdf#VPC_Scenario2

17. What is IAM? Data retrieved on 7th November 2022 from https://docs.aws.amazon.com/pdfs/IAM/latest/UserGuide/iam-ug.pdf#introduction

18. MERN Stack Explained. Data retrieved on 30th October 2022 from https://www.mongodb.com/mern-stack

19. What is Express.js? Data retrieved on 30th October 2022 from https://www.codecademy.com/article/what-is-express-js

20. Express/Node introduction. Data retrieved on 30th October 2022 from https://medium.datadriveninvestor.com/the-node-js-architecture-f86e2337bcd2

21. React. Data retrieved on 30th October 2022 from https://reactjs.org

22. Writing middleware for use in Express apps. Data retrieved on 30th October 2022 from https://expressjs.com/en/guide/writing-middleware.html

23. The Case for Enterprise-Ready Virtual Private Clouds. Data retrieved on 3rd November 2022 from https://www.usenix.org/legacy/events/hotcloud09/tech/full_papers/wood.pdf

24. Tag your Amazon EC2 resources. Data retrieved on 4th November 2022 from https://docs.aws.amazon.com/pdfs/AWSEC2/latest/UserGuide/ec2-ug.pdf#Using_Tags

25. Launch your instance. Data retrieved on 1st October 2022. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/LaunchingAndUsingInstances.html

26. Amazon EC2 security groups for Linux instances. Data retrieved on 1st October 2022 from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security-groups.html

27. Amazon EC2 key pairs and Linux instances. Data retrieved on 21st October 2022 from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html

28. Building Resilient and Fault Tolerant Applications with Micro. Data retrieved on 4th October 2022 from https://micro.dev/blog/2016/05/15/resiliency.html

29. Amazon Machine Images (AMI). Data retrieved on 11th October 2022 from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html

30. Amazon EC2 key pairs and Linux instances. Data retrieved on 11th October 2022 from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html

31. Default custom security groups. Data retrieved on 1st November 2022 from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/default-custom-security-

groups.html

32. What is NGINX? Data retrieved on 4th November 2022 from https://www.nginx.com/resources/glossary/nginx/#:~:text=NGINX%20is%20open%20source%20software,for%20maximum%20performance%20and%20stability.

33. What is a Reverse Proxy Server? Data retrieved on 12th October 2022 from https://www.nginx.com/resources/glossary/reverse-proxy-server/

34. A Complete Guide to Node.js Process Management with PM2. Data retrieved on 12th October 2022 from https://blog.appsignal.com/2022/03/09/a-complete-guide-to-nodejs-process-management-with-pm2.html

35. Amazon EC2 security groups for Linux instances. Data retrieved on 13th October 2022 from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security-groups.html

36. Yevgeiny, B. (2019).Terraform: Up & Running(3rd ed.).How Does Terraform Work? : Large Community Versus Small Community. Data retrieved on 1st November 2022 from https://www.oreilly.com/library/view/terraform-up-and/9781098116736/ch01.html#8e64a7f0-6e4b-4b3e-90f6-6b7d9d4f28d5

37. What is infrastructure as Code with Terraform? Data retrieved on 17th October 2022 from https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code

38. Terraform use cases. Data retrieved on 18th October 2022 from https://developer.hashicorp.com/terraform/intro/use-cases

39. AWS whitepaper. Data retrieved on 21st October 2022 from https://d1.awsstatic.com/whitepapers/aws-overview.pdf

40. What is AWS Lambda? Data retrieved on 1st November 2022 from https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

41. Cloud business model framework (Weinhardt et al., 2009). Data retrieved on 8th November 2022 from https://www.researchgate.net/figure/Cloud-business-model-framework-Weinhardt-et-al-2009_fig4_262451962

```
terraform {
 required_providers {
  aws = {
   source  = "hashicorp/aws"
   version = "4.34.0"
  }
 }
}
provider "aws" {
 region = "us-east-1"
}


# 1. This code block will Create VPC
resource "aws_vpc" "prod-vpc" {
 cidr_block = "10.0.0.0/16"
 tags = {
  "Name" = "Production"
 }
}
# 2. This code block will create an Internet Gateway
resource "aws_internet_gateway" "gw" {
 vpc_id = aws_vpc.prod-vpc.id
}
```

```
# 3. It will Create a custom route table
resource "aws_route_table" "prod-route-table" {
 vpc_id = aws_vpc.prod-vpc.id

 route {
  cidr_block = "0.0.0.0/0"
  gateway_id = aws_internet_gateway.gw.id
 }
 route {
  ipv6_cidr_block = "::/0"
  gateway_id      = aws_internet_gateway.gw.id
 }
 tags = {
  Name = "Production"
 }
}
# 4. Create a subnet for the VPC
resource "aws_subnet" "subnet-1" {
 vpc_id          = aws_vpc.prod-vpc.id
 cidr_block      = "10.0.0.0/16"
 availability_zone = "us-east-1a"
 tags = {
  "Name" = "Production-subnet"
 }
```

```
}

# 5.This code will make an Associate subnet with route table
resource "aws_route_table_association" "a" {
 subnet_id      = aws_subnet.subnet-1.id
 route_table_id = aws_route_table.prod-route-table.id
}

# 6.Create a security group to allow the following PORT (22,80,443)
resource "aws_security_group" "allow_web" {
 name        = "allow_web_traffic"
 description = "Allow web inbound traffic"
 vpc_id      = aws_vpc.prod-vpc.id

 ingress {
  description = "HTTPS"
  from_port   = 443
  to_port     = 443
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
 }

 ingress {
  description = "SSH"
  from_port   = 22
  to_port     = 22
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}
ingress {
  description = "HTTP"
  from_port   = 80
  to_port     = 80
  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"]

}

tags = {
  Name = "allow_WEB"
```

```
 }
}


# 7.Create a network interface with an IP that was created in step
5.
resource "aws_network_interface" "web-server-net-interface" {
 subnet_id      = aws_subnet.subnet-1.id
 private_ips    = ["10.0.1.50"]
 security_groups = [aws_security_group.allow_web.id]
}
# 8.Assign an EIP & attached to the Internet Gateway
resource "aws_eip" "server-eip" {
 vpc                 = true
 network_interface       = aws_network_interface.web-server-net-
interface.id
 associate_with_private_ip = "10.0.1.50"
 depends_on           = [aws_internet_gateway.gw]
 tags = {
  Name = "serverEIP"
 }


}


# Create an Ubuntu server and install/enable apache 2.
```

```
resource "aws_instance" "Thesis_server" {
 ami           = "ami-08c40ec9ead489470"
 instance_type    = "t2.micro"
 availability_zone = "us-east-1a"
 key_name       = "web-server-key-pair"

 network_interface {
  device_index      = 0
  network_interface_id = aws_network_interface.web-server-net-interface.id
 }

 user_data = <<-EOF
        #!/bin/bash
         sudo apt update -y
         sudo apt install apache2 -y
         sudo start apache2
         sudo bash -c "echo Hello From Oamk > /var/www/html/index.html"
        EOF
 tags = {
  Name = "Thesis Server"
 }
}
```