A

**REPORT**

Of

**MINI PROJECT OR INTERNSHIP ASSESSMENT**

At

**“LinuxWorld Informatics Pvt Ltd, Jaipur”**



On

**“DevOps With Cloud”**

Submitted

In the partial fulfilment of Bachelor of Technologies

**Department of Computer Science and Engineering**



**Shri Ram Murti Smarak College of Engineering and Technologies, Bareilly**

(2025-26)

Submitted To Submitted By

Dr. Juhi Chaudhary Yash Gupta

Assistant Professor(CSE) (2200140100123)

# Certificate of Internship



# Acknowledgement

I would like to express my profound gratitude to everyone who supported me throughout the course of this internship and in the completion of this project report. This journey has been an invaluable learning experience, and it would not have been possible without the guidance and support of many individuals.

Primarily, I extend my deepest thanks to my mentor at LinuxWorld Informatics Pvt. Ltd., **Mr. Vimal Daga**. His unparalleled expertise, practical teaching style, and constant encouragement were the driving forces behind my learning. His ability to break down complex technologies into understandable, real-world examples has been truly inspiring and has provided me with a solid foundation in DevOps and Cloud Computing.1 I am also grateful to the entire team at LinuxWorld for providing such a conducive and professional learning environment.

I would also like to thank my university, and the faculty of the Department for providing me with this opportunity to engage in an industrial internship. The theoretical knowledge gained in my coursework was the bedrock upon which the practical skills of this internship were built.

Finally, I owe my heartfelt thanks to my family and friends for their unwavering support, patience, and belief in me. Their encouragement was a constant source of motivation during this intensive program.

This report is a culmination of the knowledge and skills I have acquired, and I hope it reflects the excellent training and guidance I have received.

**Yash Gupta**

**Table Of Content**

[Certificate of Internship ii](#_Toc207875482)

[Acknowledgement iii](#_Toc207875483)

[List of Figures vi](#_Toc207875484)

[List of Tables vii](#_Toc207875485)

[Chapter 1 1](#_Toc207875486)

[Company Profile: LinuxWorld Informatics Pvt. Ltd. 1](#_Toc207875487)

[1.1. Introduction and History 1](#_Toc207875488)

[1.2. Vision, Mission, and Commitment to Open Source 1](#_Toc207875489)

[1.3. Core Services and Technological Focus 2](#_Toc207875490)

[1.4. Strategic Partnership with Red Hat 3](#_Toc207875491)

[Chapter 2 5](#_Toc207875492)

[Industry at a Glance: The DevOps and Cloud Revolution 5](#_Toc207875493)

[2.1. The Evolution of IT Infrastructure 5](#_Toc207875494)

[2.2. The DevOps Movement: A Cultural and Technical Paradigm Shift 6](#_Toc207875495)

[2.3. Market Analysis: Cloud and DevOps Adoption in India and Globally 8](#_Toc207875496)

[Chapter 3 11](#_Toc207875497)

[Tools and Technology Used in Industry 11](#_Toc207875498)

[3.1. Cloud Platform: Amazon Web Services (AWS) 11](#_Toc207875499)

[3.2. Infrastructure as Code: Terraform 12](#_Toc207875500)

[3.3. Configuration Management: Ansible 12](#_Toc207875501)

[3.4. Containerization and Orchestration: Docker & Kubernetes 13](#_Toc207875502)

[3.5. CI/CD and Automation: Git, GitHub, & Jenkins 13](#_Toc207875503)

[3.6. Monitoring : CloudWatch, Prometheus, & Grafana 13](#_Toc207875504)

[3.7. Development and Scripting: Python, Flask, & Bash 14](#_Toc207875505)

[Chapter 4 15](#_Toc207875506)

[Modules of Industry (Projects Undertaken) 15](#_Toc207875507)

[4.1. Project I: Automated Provisioning of Cloud Infrastructure for a Web Application 15](#_Toc207875508)

[4.2. Project II: Declarative Configuration of a Multi-Server Web Environment 15](#_Toc207875509)

[4.3. Project III: Architecting for High Availability and Scalability 15](#_Toc207875510)

[4.4. Project IV: End-to-End CI/CD Pipeline for a Containerized API 16](#_Toc207875511)

[4.5. Project V: Research and Application of Agentic AI in DevOps 16](#_Toc207875512)

[Chapter 5 18](#_Toc207875513)

[Details about the Modules (Implementation and Analysis) 18](#_Toc207875514)

[5.1. Project I Walkthrough: Provisioning AWS Infrastructure with Terraform 18](#_Toc207875515)

[5.1.1. Prerequisites and Setup 18](#_Toc207875516)

[5.1.2. Writing the Terraform Configuration 19](#_Toc207875517)

[1. Provider and Version (terraform.tf) 19](#_Toc207875518)

[2. Main Infrastructure (main.tf) 19](#_Toc207875519)

[5.1.3. The Terraform Workflow in Practice 21](#_Toc207875520)

[5.1.4. Advanced Concepts for Production Environments 21](#_Toc207875521)

[5.2. Project II Walkthrough: Configuring Web Servers with Ansible Playbooks 23](#_Toc207875522)

[5.2.1. Ansible Setup and Inventory 23](#_Toc207875523)

[5.2.2. From Ad-hoc Commands to a Declarative Playbook 23](#_Toc207875524)

[5.2.3. Advanced Concepts for Scalable and Secure Automation 26](#_Toc207875525)

[5.3. Project III Walkthrough: Implementing ELB and Auto Scaling Groups 28](#_Toc207875526)

[5.3.1. Architecture Design 28](#_Toc207875527)

[5.3.2. Terraform Implementation 28](#_Toc207875528)

[5.3.3. Demonstration and Analysis 31](#_Toc207875529)

[5.4. Project IV Walkthrough: Building the Automated Jenkins CI/CD Pipeline 33](#_Toc207875530)

[5.4.1. The Application and its Tests 33](#_Toc207875531)

[5.4.2. Dockerizing the Application with Best Practices 34](#_Toc207875532)

[5.4.3. The Jenkins CI/CD Pipeline (Jenkinsfile) 36](#_Toc207875533)

[5.5. Project V Analysis: A Practical Exploration of GenAI Ops 38](#_Toc207875534)

[5.5.1. Core Concepts: Agentic AI and Chain of Thought 38](#_Toc207875535)

[5.5.2. Practical Demonstration in a Jupyter Notebook 38](#_Toc207875536)

[5.5.3. The Future Role of the DevOps Engineer 40](#_Toc207875537)

[Chapter 6 42](#_Toc207875538)

[Training Outcome 42](#_Toc207875539)

[6.1. Summary of Acquired Technical Competencies 42](#_Toc207875540)

[6.2. Application of DevOps Principles in Practice 43](#_Toc207875541)

[6.3. Challenges Encountered and Problem-Solving Strategies 43](#_Toc207875542)

[6.4. Personal and Professional Growth 45](#_Toc207875543)

[Chapter 7 46](#_Toc207875544)

[Conclusion 46](#_Toc207875545)

[7.1. Synthesis of Project Achievements 46](#_Toc207875546)

[7.2. The Business Impact of an Integrated DevOps and Cloud Strategy 47](#_Toc207875547)

[7.3. Future Scope and Recommended Enhancements 48](#_Toc207875548)

[References 50](#_Toc207875549)

# List of Figures

# List of Tables

[Table 2. 1 India Cloud and DevOps Market Projections (2025-2031)18 20](#_Toc207611736)

[Table 5. 1: Key Terraform Commands and Their Functions 32](#_Toc207611755)

[Table 5. 2: Common Ansible Modules Used in Web Server Configuration 36](#_Toc207611756)

[Table 5. 3: Jenkins Pipeline Stages and Their Functions 46](#_Toc207611757)

# 

# Chapter 1

# Company Profile: LinuxWorld Informatics Pvt. Ltd.

## 1.1. Introduction and History

LinuxWorld Informatics Pvt. Ltd., commonly known as 'LW', is a distinguished technology and training solutions company headquartered in Jaipur, Rajasthan.2 Established in 2005, the organization has consistently positioned itself at the forefront of the open-source technology movement in India. It is an ISO 9001:2008 Certified Organisation, a testament to its commitment to quality and standardized processes.1

The company was formally incorporated as a private limited entity on June 12, 2012, under the dynamic leadership of its directors, including the visionary technologist Mr. Vimal Daga.3 From its inception, LinuxWorld has been governed by a team of young and energetic technocrats dedicated to the promotion of Linux and open-source technologies.1 Over nearly two decades, it has evolved into a center of excellence, recognized for its state-of-the-art infrastructure, innovative training methodologies, and a customer-centric approach that has been readily accepted by organizations across India.2 The company's longevity and formal structure provide a credible and stable foundation for the high-end industrial training programs it offers, including the 'DevOps with Cloud' internship.

## 1.2. Vision, Mission, and Commitment to Open Source

At its core, LinuxWorld is driven by a profound and ambitious vision: "To come out with the best operating system i.e. a globally acceptable product this would be different, new and useful to the entire world".1 This forward-looking goal informs the company's mission, which is to establish itself as a single-source provider of premier learning and technology solutions. The organization aims to empower companies and individuals by enhancing their technological capabilities and preparing their workforce for the evolving demands of the IT industry.2

A defining characteristic of LinuxWorld is its unwavering commitment to the principles of open-source software. This philosophy is not merely a business strategy but the central pillar of its identity. The company actively promotes Linux and associated open-source technologies through its training and services.1 This commitment is physically manifested in its dedicated Research & Development Centre for the Linux Operating System, a facility reportedly unique in India. The R&D team's objective is not just to edit existing software but to innovate by identifying and resolving loopholes in the Linux OS and developing novel tools and functionalities.1

This deep-rooted ideology directly shapes the educational experience provided to interns. The curriculum for the 'DevOps with Cloud' internship is a direct reflection of this corporate philosophy. The technologies covered—Linux, Git, Jenkins, Docker, Kubernetes, Terraform, and Ansible—are overwhelmingly open-source projects.5 This intentional selection demonstrates that the training is not just a random assortment of popular tools but a curated immersion into the collaborative, non-proprietary technological ecosystem that is at the heart of the DevOps movement itself. The internship, therefore, represents an opportunity to learn not just the "how" of the tools, but the "why" behind the open-source culture that created them.

## 1.3. Core Services and Technological Focus

LinuxWorld offers an integrated and comprehensive portfolio of services that cater to a wide spectrum of IT needs, solidifying its reputation as a full-service technology solutions company. These services can be broadly categorized into training and development, technical support, and research and development.2

The company's training division is its most prominent facet, providing industry-relevant education on a vast array of cutting-edge technologies. The course catalog includes specializations in:

* **DevOps:** Covering tools like Ansible, Jenkins, Git, Docker, Kubernetes, Prometheus, and Grafana.6
* **Cloud Computing:** With a strong focus on Amazon Web Services (AWS) and overviews of Microsoft Azure and Google Cloud Platform (GCP).6
* **Data Science and AI:** Including Machine Learning, Deep Learning, and NLP.6
* **Full Stack Development:** Focusing on the MERN (MongoDB, Express.js, React, Node.js) stack.6
* **Networking and Security:** Offering globally recognized certifications like CCNA, CCNP, and Ethical Hacking.1

This extensive expertise has attracted over 100,000 students and 35,000 working professionals, earning LinuxWorld the distinction of being awarded the #1 company for internships across India and the Asia Pacific region.6 The 'Next Generation DevOps With Cloud Computing' project track, which forms the basis of this internship, is a flagship program that synthesizes the company's core competencies in cloud and automation.9

Beyond training, LinuxWorld provides high-end technical support services for all major open-source applications, positioning itself as a reliable partner for enterprises leveraging this software.1 This practical, real-world support experience feeds back into the training curriculum, ensuring that the knowledge imparted is not just theoretical but grounded in the challenges and solutions of live production environments.

## 1.4. Strategic Partnership with Red Hat

A cornerstone of LinuxWorld's credibility and a significant value-add to its training programs is its strategic partnership with Red Hat. Red Hat is the world's leading provider of enterprise open-source solutions, renowned for its highly secure and stable Red Hat Enterprise Linux (RHEL) operating system and the powerful Ansible Automation Platform.1LinuxWorld has been officially recognized by Red Hat as a "Most Promising Partner," signifying a deep and effective collaboration.1

This partnership manifests in several key ways:

* **Curriculum Alignment:** The training curriculum at LinuxWorld is closely aligned with official Red Hat courses and certifications. For instance, the internship covers material relevant to the Red Hat Certified System Administrator (RHCSA) and the Red Hat Certified Engineer in Ansible Automation (EX294) exams.5
* **Enterprise-Grade Focus:** The collaboration ensures that the training goes beyond the open-source project versions of tools and focuses on their enterprise-grade, commercially supported counterparts. Learning Ansible within the context of the Red Hat Ansible Automation Platform (AAP), for example, prepares interns for the realities of large-scale, production deployments where stability, security, and support are paramount.5
* **Industry Recognition and Talent Pipeline:** The quality of training is validated by the fact that Red Hat frequently hires graduates from LinuxWorld's programs, viewing the institution as a reliable source of skilled talent.11
* **Expert Collaboration:**LinuxWorld frequently hosts joint expert sessions with senior specialists from Red Hat, providing interns with direct access to industry leaders and deep-dive knowledge on advanced topics like Ansible Tower and automation best practices.12

This strong Red Hat partnership ensures that the skills acquired during the internship are not purely academic. They are enterprise-ready and immediately applicable in corporate IT environments. The training provides an understanding not just of how a tool works, but how it is deployed, managed, and secured within the rigorous standards of a production ecosystem, making the learning experience immensely valuable for career development.

# Chapter 2

# Industry at a Glance: The DevOps and Cloud Revolution

## 2.1. The Evolution of IT Infrastructure

To fully appreciate the significance of modern DevOps practices, it is essential to understand the evolutionary journey of IT infrastructure. This history reveals a continuous search for greater efficiency, scalability, and agility, culminating in the cloud computing paradigm that underpins today's digital world.

**The Mainframe and Minicomputer Era (1959 - 1980s):** The dawn of commercial computing was dominated by large, centralized mainframe computers, such as the IBM 360 series.13 These powerful machines served entire organizations, with users accessing them through remote terminals in a time-sharing model. While revolutionary, this approach was characterized by enormous capital expenditure, specialized operational staff, and a highly rigid, centralized control structure. The subsequent rise of minicomputers offered a more decentralized model, but the fundamental principle of owning and managing physical hardware remained.13

**The Personal Computer and Client/Server Era (1980s - 2000s):** The proliferation of the personal computer (PC) and the development of client/server architecture marked a significant shift towards distributed computing.13 In this model, applications were split between client machines (desktops) and more powerful servers connected over a local network. This brought computing power directly to users and departments, promoting productivity. However, it also introduced new complexities: managing vast fleets of PCs, ensuring network connectivity, and dealing with inconsistent software environments across machines. Server provisioning was still a manual, time-consuming process involving physical hardware procurement, installation, and configuration, often taking weeks or months.5

**The Rise of Cloud Computing (2000s - Present):** The limitations of the client/server model—slow provisioning, high upfront costs, and operational overhead—paved the way for the next paradigm shift: Cloud Computing. Pioneered and popularized by providers like Amazon Web Services (AWS), cloud computing offered a revolutionary model: the on-demand delivery of IT resources over the internet with pay-as-you-go pricing.5

This new model abstracted the underlying physical hardware, offering it to users through service models:

* **Infrastructure as a Service (IaaS):** Provides fundamental computing resources like virtual servers (e.g., AWS EC2), storage (e.g., AWS S3), and networking (e.g., AWS VPC). Users manage the operating system and applications, but not the physical hardware.5 This is the foundational layer for most DevOps work.
* **Platform as a Service (PaaS):** Offers a platform for developers to build and deploy applications without managing the underlying infrastructure (e.g., AWS Elastic Beanstalk).5
* **Software as a Service (SaaS):** Delivers ready-to-use software applications over the internet (e.g., Gmail, Salesforce).5

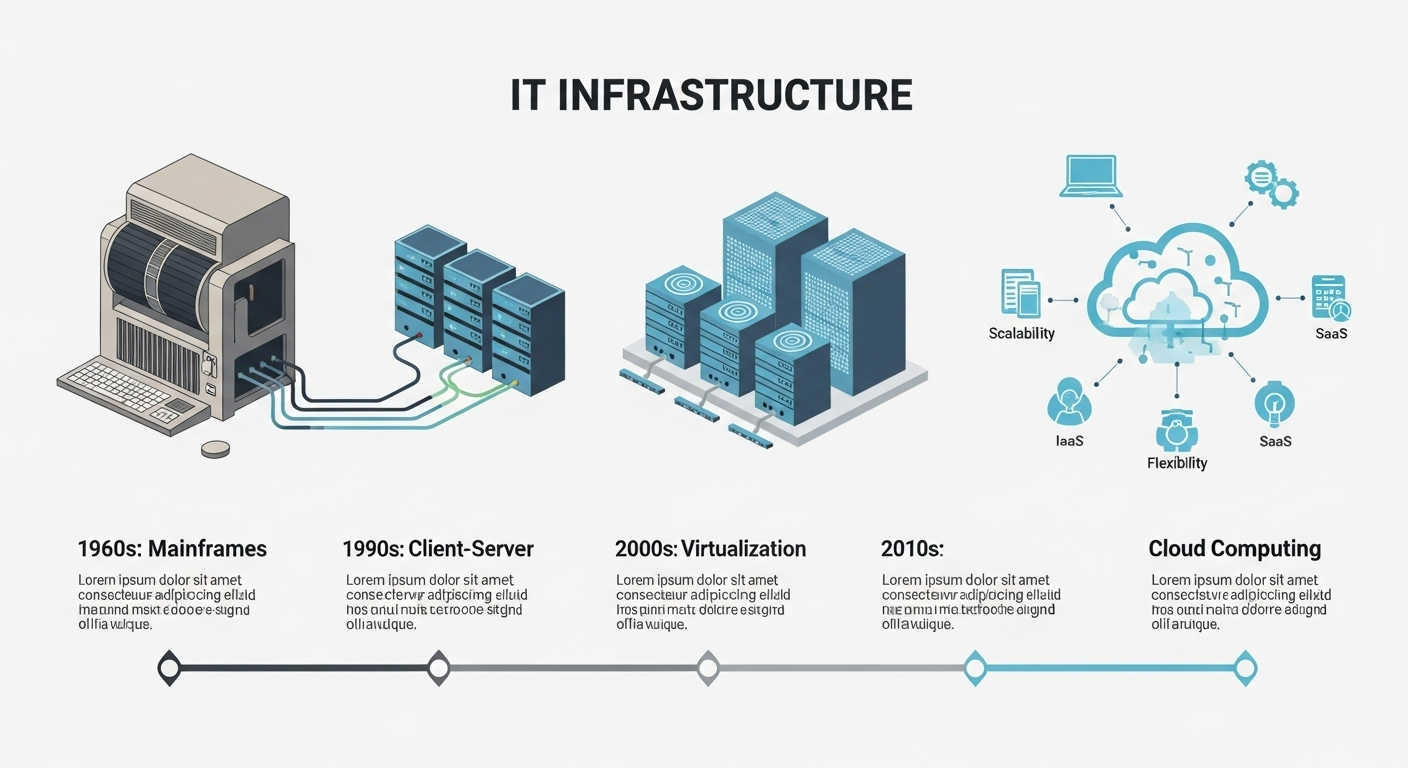
The most crucial innovation of the cloud was not just virtualization, but the exposure of infrastructure management through Application Programming Interfaces (APIs). This meant that for the first time, infrastructure could be provisioned, configured, and managed programmatically, setting the stage for the automation revolution of DevOps.5

Figure 2. 1 IT Infrastructure

## 2.2. The DevOps Movement: A Cultural and Technical Paradigm Shift

The emergence of API-driven cloud infrastructure created the perfect environment for a new methodology to flourish: DevOps. DevOps is not merely a tool or a technology, but a cultural and professional movement that aims to break down the traditional silos between software development (Dev) and IT operations (Ops) teams.5 Its primary goal is to increase software delivery velocity, improve service reliability, and build shared ownership among all stakeholders.

The term was popularized following the first "DevOps Days" conference in Belgium in 2009, born from the need to align the agile, iterative practices of software development with the operational realities of deployment and maintenance.16 Traditional models saw developers "throwing code over the wall" to operations teams, leading to friction, delays, and miscommunication. DevOps seeks to merge these functions into a single, streamlined workflow.

The philosophy of DevOps is often encapsulated in two key frameworks:

1. **The Three Ways:**
   * **The First Way (Flow):** Emphasizes accelerating the flow of work from development to operations to the customer. This is achieved by minimizing batch sizes, automating processes, and removing bottlenecks.15
   * **The Second Way (Feedback):** Focuses on creating rapid and constant feedback loops from right to left (from operations back to development). This allows for early detection of problems, continuous improvement, and a better understanding of system performance.15
   * **The Third Way (Continuous Learning and Experimentation):** Fosters a high-trust culture that encourages experimentation, risk-taking, and learning from failure. This drives innovation and builds organizational resilience.15
2. **The CALMS Framework:**
   * **C**ulture: Fostering collaboration, shared responsibility, and trust.
   * **A**utomation: Automating repetitive tasks to reduce human error and increase speed.
   * **L**ean: Applying lean manufacturing principles to eliminate waste and optimize the value stream.
   * **M**easurement: Collecting data and metrics on all parts of the delivery process to drive improvement.
   * **S**haring: Ensuring knowledge and tools are shared across teams.15

A fundamental tenet that enables these principles is the concept of **"Everything as Code"**.5 By defining infrastructure (Infrastructure as Code), configurations (Configuration as Code), and delivery pipelines (Pipeline as Code) in version-controlled text files, the entire software delivery process becomes automated, repeatable, and transparent.

The relationship between cloud computing and DevOps is not just complementary; it is a symbiotic and mutually reinforcing growth cycle. The on-demand, API-driven nature of the cloud made DevOps automation practices technically feasible. In turn, the adoption of DevOps practices, which enable organizations to manage infrastructure at scale, has driven the exponential growth and consumption of cloud services. One cannot be fully realized without the other.

## 2.3. Market Analysis: Cloud and DevOps Adoption in India and Globally

The skills and technologies acquired during this internship are positioned at the epicenter of one of the most significant growth sectors in the global and Indian economies. The demand for cloud and DevOps expertise is not a fleeting trend but a sustained, long-term market reality driven by the fundamental need for businesses to become more agile, efficient, and innovative.

**Global Market Trajectory:** The global DevOps market is experiencing explosive growth. Valued at approximately $10.4 billion in 2023, it is projected to surge to $25.5 billion by 2028, expanding at a compound annual growth rate (CAGR) of 19.7%.17 This rapid expansion reflects the widespread understanding that DevOps practices are no longer a niche advantage but a core business imperative for competitive software delivery.

**Figure 2.2** Market Cap for Devops and Cloud

**The Indian Growth Story:** The Indian market mirrors this global trend, with its own unique growth catalysts.

* **DevOps Market in India:** The market was valued at $3.81 billion in 2025 and is forecast to reach $10.80 billion by 2031, demonstrating a robust CAGR of 18.96%.18
* **Cloud Computing Market in India:** The cloud adoption growth is steeper. The Indian cloud computing market is estimated to grow from $21.82 billion in 2025 to an impressive $58.73 billion by 2030, at a CAGR of 21.90%.19

This accelerated adoption is fueled by several factors, including a strong push from government initiatives like "Digital India," a widespread digital transformation surge across all enterprise sectors, and massive investments in hyperscale data centers by major cloud providers like AWS, Microsoft, and Google.19 AWS, in particular, has a dominant presence, with its Indian net sales growing by a remarkable 43% in fiscal year 2023 and a commitment to invest over $12.7 billion in the country's infrastructure by 2030.21

Table 2. 1 India Cloud and DevOps Market Projections (2025-2031)18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Market Segment | 2025 Market Size (USD) | Projected Market Size (USD) | Forecast Period | CAGR (%) |
| India DevOps Market | $3.81 Billion | $10.80 Billion | 2026 - 2031 | 18.96% |
| Indian Cloud Market | $ 28 Billion | $ 58 Billion | 2026 - 2031 | 21.5% |

This explosive growth has created a direct and critical market need. While demand for skilled professionals is soaring, the supply has struggled to keep pace, resulting in a significant skills gap. A NASSCOM report highlights this disparity, estimating that India will require over 2 million cloud professionals by 2025 but is on track to produce only 1.5 million with baseline growth.22 This creates a disequilibrium in the talent market.

Training organizations like LinuxWorld exist precisely to bridge this gap.2 The intensive, practical, and industry-aligned internship program is a strategic response to this market demand. By completing this internship, I have acquired a skill set that is not just technically advanced but also in critically short supply, positioning me as a highly valuable asset in one of India's fastest-growing and most vital economic sectors. This report, therefore, documents not just a period of learning, but a strategic step towards a promising career.

# Chapter 3

# Tools and Technology Used in Industry

The 'DevOps with Cloud' internship provided a comprehensive, hands-on immersion into the modern technology stack that powers automated, scalable, and resilient software delivery. This chapter provides a high-level overview of the key tools and technologies I worked with, outlining their specific roles within the DevOps lifecycle. The detailed implementation of these tools will be covered in Chapter 5.

## 3.1. Cloud Platform: Amazon Web Services (AWS)

Amazon Web Services (AWS) served as the foundational cloud platform for all project work. As the industry's leading Infrastructure as a Service (IaaS) provider, AWS offers a vast and mature ecosystem of on-demand services that enable the creation of complex infrastructure programmatically.23 My work focused on its core components, which are the building blocks of any cloud-native application:

* **Compute:** Amazon EC2 (Elastic Compute Cloud) for providing resizable virtual servers (instances).
* **Networking:** Amazon VPC (Virtual Private Cloud) for creating logically isolated and secure network environments.
* **Storage:** Amazon S3 (Simple Storage Service) for object storage and EBS (Elastic Block Store) for persistent block storage for EC2 instances.
* **Identity & Access Management (IAM):** For securely managing access to AWS services and resources.
* **Monitoring:** Amazon CloudWatch for collecting metrics, logs, and events from AWS resources.5



Figure 3. 1 Amazon Web Services

## 3.2. Infrastructure as Code: Terraform

Terraform by HashiCorp was the primary tool used for Infrastructure as Code (IaC). It is a declarative, open-source tool that allows infrastructure to be defined, provisioned, and managed using human-readable configuration files.5 The key purpose of using Terraform was to automate the creation of the entire AWS environment, from networking to compute instances. Its vendor-neutral nature means that the same principles and workflows can be applied to other cloud providers, making it a highly transferable skill.5

## 3.3. Configuration Management: Ansible

Once infrastructure was provisioned by Terraform, Ansible was used for configuration management. Ansible is a powerful, agentless, and declarative automation engine designed to automate software installation, system configuration, and application deployment on existing servers.5 Its agentless architecture, communicating over standard SSH, simplifies setup and management. I used Ansible to transform bare EC2 instances into fully functional web servers, ensuring that configurations were consistent, repeatable, and idempotent (meaning the same operation could be run multiple times without unintended side effects).5

## 3.4. Containerization and Orchestration: Docker & Kubernetes

Docker was the chosen technology for containerization. It solves the classic "it works on my machine" problem by packaging an application and all its dependencies into a lightweight, portable unit called a container.5 This ensures the application runs consistently across any environment, from a developer's laptop to a production server. I used Docker to containerize a Python web application. While the project focused on Docker, the curriculum also introduced Kubernetes as the industry-standard open-source platform for automating the deployment, scaling, and management of containerized applications at scale.5

## 3.5. CI/CD and Automation: Git, GitHub, & Jenkins

The foundation of any automated pipeline is version control. I used **Git**, the industry-standard distributed version control system, to track changes in all code, including application code, Terraform configurations, and Ansible playbooks. **GitHub** was used as the centralized, cloud-based repository for hosting the Git projects.5

**Jenkins** was the orchestrator at the heart of the Continuous Integration/Continuous Deployment (CI/CD) pipeline. As a leading open-source automation server, Jenkins was configured to connect to GitHub, automatically detect code changes, and trigger a series of automated steps including testing, building a Docker image, and preparing the application for deployment.5

## 3.6. Monitoring : CloudWatch, Prometheus, & Grafana

To ensure the health and performance of the deployed infrastructure and applications, a multi-faceted monitoring strategy was explored. **AWS CloudWatch** was used for its native integration with AWS services, providing essential metrics like CPU utilization, network traffic, and logs.5 The training also introduced the powerful open-source duo of

**Prometheus** and **Grafana**. Prometheus is a time-series database and monitoring system designed for reliability and scalability, while Grafana is a leading platform for creating rich, interactive, and insightful visualization dashboards from various data sources, including Prometheus and CloudWatch.5

## 3.7. Development and Scripting: Python, Flask, & Bash

A strong foundation in scripting and development is crucial for a DevOps engineer. **Python** was used as the primary language for this purpose due to its versatility, readability, and extensive libraries. Specifically, the **Flask** framework was used to build a lightweight web API for the CI/CD project.5 In addition,

**Bash scripting** was utilized for essential Linux system administration tasks and for writing simple automation scripts within the Jenkins pipeline, demonstrating the ability to work directly with the underlying operating system.5

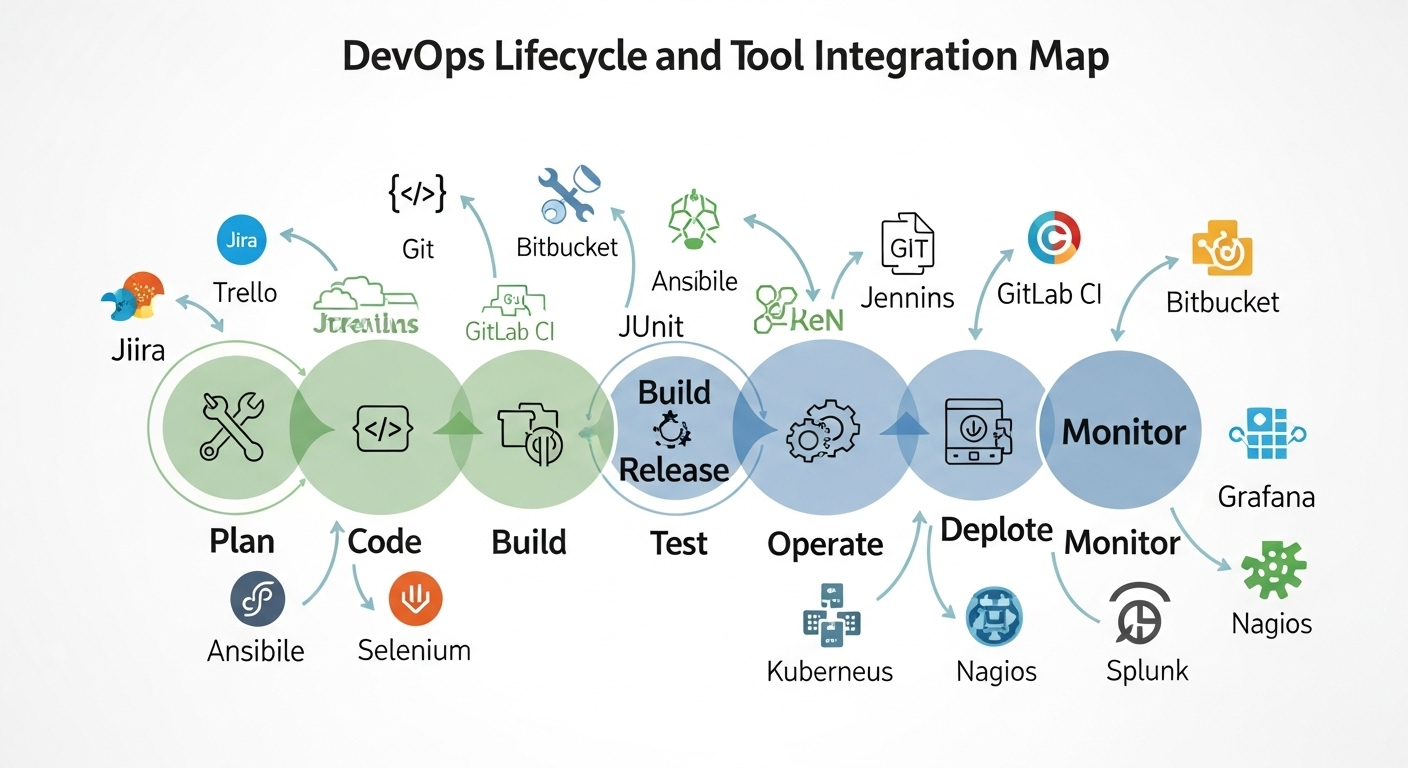


Figure 3. 2 Devops Lifecycle and Tool Integration Map

# Chapter 4

# Modules of Industry (Projects Undertaken)

This chapter outlines the high-level objectives of the five core project modules I undertook during the internship. These projects were designed to be cumulative, with each module building upon the skills and infrastructure of the previous one. This structure provided a practical, step-by-step journey from foundational concepts to a fully integrated, automated system. The detailed implementation, analysis, and code for each module are presented in Chapter 5.

## 4.1. Project I: Automated Provisioning of Cloud Infrastructure for a Web Application

**Objective:** To master the principles of Infrastructure as Code (IaC) by using Terraform to fully automate the provisioning of a foundational cloud environment on Amazon Web Services. This module focused on moving away from manual, click-based console operations to a repeatable, version-controlled, and code-driven approach for creating core infrastructure components. The key outcome was a script that could create a complete, functional, and isolated network and compute environment from scratch.5

## 4.2. Project II: Declarative Configuration of a Multi-Server Web Environment

**Objective:** To apply the principles of declarative configuration management using Ansible to automate the setup of software on the infrastructure provisioned in Project I. This module aimed to demonstrate how to transform a bare operating system into a fully configured web server without manual intervention. The focus was on ensuring consistency, reliability, and idempotency, where the same configuration script could be run repeatedly to enforce a desired state across multiple servers.5

## 4.3. Project III: Architecting for High Availability and Scalability

**Objective:** To design and implement a cloud-native architecture capable of withstanding component failure and dynamically scaling to meet fluctuating user demand. This module involved integrating advanced AWS services to build upon the previous projects. The goal was to create a resilient, self-healing, and cost-efficient web application environment that could automatically distribute traffic and adjust its compute capacity, which are critical requirements for any production-grade application.5

## 4.4. Project IV: End-to-End CI/CD Pipeline for a Containerized API

**Objective:** To orchestrate a complete Continuous Integration and Continuous Deployment (CI/CD) pipeline using Jenkins. This capstone project synthesized skills from across the DevOps spectrum. The goal was to create a fully automated workflow that would take a developer's code commit from a GitHub repository and automatically build, test, containerize, and prepare the application for deployment. This module demonstrated the core value proposition of DevOps: enabling rapid, reliable, and frequent software releases.5

## 4.5. Project V: Research and Application of Agentic AI in DevOps

**Objective:** To explore the future of IT automation by researching the emerging field of Generative AI Operations (GenAI Ops). This module focused on understanding the concepts behind Agentic AI, where Large Language Models (LLMs) act as reasoning engines to perform complex tasks. The practical goal was to demonstrate how an AI agent could be prompted with natural language to generate functional code for DevOps tasks, such as creating Terraform configurations, showcasing an understanding of the next frontier in automation.5

Figure 4. 1 Overview of Industry Module

# Chapter 5

# Details about the Modules (Implementation and Analysis)

This chapter provides a comprehensive, in-depth walkthrough of each project module outlined in Chapter 4. It includes detailed technical explanations, step-by-step implementation guides, relevant code snippets, and an analysis of the concepts and best practices applied.

## 5.1. Project I Walkthrough: Provisioning AWS Infrastructure with Terraform

This project laid the groundwork for all subsequent modules by automating the creation of a secure and functional AWS environment using Terraform. The focus was on applying Infrastructure as Code (IaC) principles to ensure the infrastructure was versionable, reusable, and consistently reproducible.

### 5.1.1. Prerequisites and Setup

Before writing any Terraform code, the local environment and AWS account must be prepared for programmatic access.

1. **AWS CLI Installation:** The AWS Command Line Interface was installed to enable interaction with AWS services from the terminal.
2. **IAM User Creation:** An IAM (Identity and Access Management) user was created in the AWS console. Crucially, this user was configured with **Programmatic access**, which generates an Access Key ID and a Secret Access Key. These credentials are used by Terraform to authenticate API requests to AWS.24 For initial setup, the  
   AdministratorAccess policy was attached to the user, though in a production scenario, a more restrictive policy following the principle of least privilege would be used.
3. **Configuring AWS Credentials:** The generated credentials were configured locally using the aws configure command. This stores the keys in a secure location that the Terraform AWS provider can automatically detect and use for authentication.24

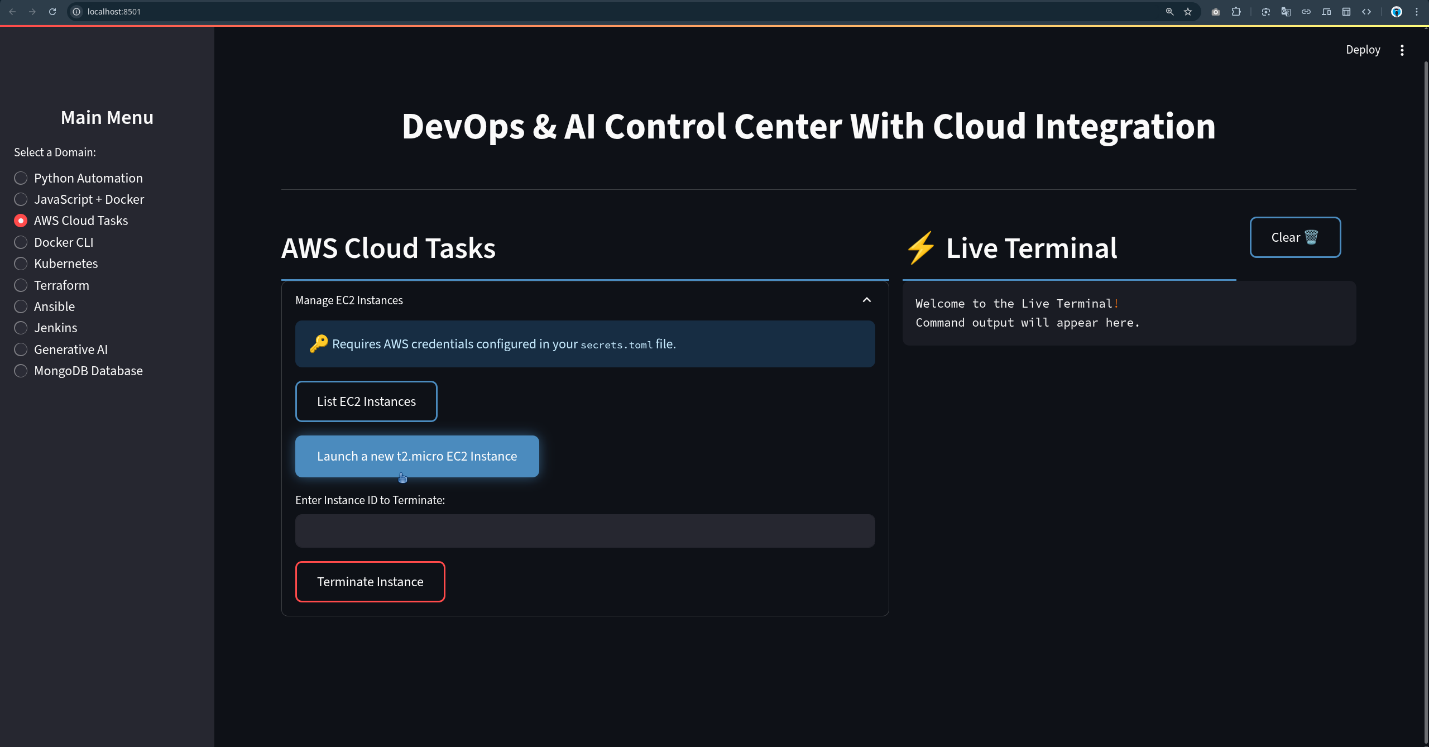


Figure 6. 1 Amazon Cloud Services Tasks

### 5.1.2. Writing the Terraform Configuration

Terraform is a tool that helps us create and manage cloud infrastructure (like servers, networks, and firewalls) automatically using code instead of clicking around in AWS.

### 1. ****Provider and Version (terraform.tf)****

* Think of a provider as the connection between Terraform and AWS.
* In this file, we tell Terraform:
  + “Use AWS as the provider.”
  + “Make sure the AWS plugin is version 5.x.”
  + “Make sure Terraform itself is at least version 1.2.0.”

This ensures everything works smoothly with the right versions.

### 2. ****Main Infrastructure (main.tf)****

This is where we describe the actual AWS resources we want.

1. **Provider Block (region setup)**
   * We choose which AWS region to create resources in. Example: us-west-2 (Oregon).
2. **VPC and Networking**
   * A **VPC (Virtual Private Cloud)** is like your own private network in AWS.
   * Inside it, we create:
     + **Subnet (Public Subnet):** A smaller section of the network where our server will live.
     + **Internet Gateway (IGW):** A doorway that allows our network to connect to the internet.
     + **Route Table:** A map that says, “Send all internet traffic through the Internet Gateway.”
     + We link this route table to the subnet.
3. **Security Group (Firewall rules)**
   * Like a virtual firewall around our server.
   * Rules:
     + Allow **SSH (port 22)** so we can log in remotely.
     + Allow **HTTP (port 80)** so people can view the website.
     + Allow all outgoing traffic (server can access internet freely).
4. **EC2 Instance (Virtual Server)**
   * This is the actual computer running in the cloud.
   * Details:
     + Uses Amazon Linux 2 (a lightweight operating system).
     + Size is **t2.micro** (cheap and small for testing).
     + Connected to our subnet and security group.
     + Uses an existing key pair so we can log in securely.



Figure 5. 2 AWS Cloud

### 5.1.3. The Terraform Workflow in Practice

With the configuration files written, I executed the standard Terraform workflow to bring the infrastructure to life.

Table 5. 1: Key Terraform Commands and Their Functions

|  |  |
| --- | --- |
| Command | Purpose |
| terraform init | Initializes the working directory. It downloads the specified AWS provider plugin and sets up the backend. This is run once per project. 5 |
| terraform validate | Checks the configuration files for syntax errors and internal consistency without accessing remote services. A crucial step for catching typos. 5 |
| terraform plan | Creates an execution plan. Terraform compares the desired state (in the code) with the current state (in the cloud) and shows exactly what it will create, modify, or destroy. This is a critical review step. 5 |
| terraform apply | Executes the plan after receiving explicit user confirmation (yes). This is the command that provisions the actual resources in AWS. 5 |
| terraform destroy | Tears down all resources managed by the configuration. This is essential for cleaning environments and avoiding costs. 5 |

Running terraform applications resulted in the successful creation of the VPC, subnet, IGW, route table, security group, and the EC2 instance, all in a matter of minutes—a process that would be significantly more time-consuming and error-prone if done manually.

### 5.1.4. Advanced Concepts for Production Environments

To demonstrate an understanding of real-world DevOps practices, I explored concepts that go beyond a simple single-user setup.

Remote State Management with S3 and DynamoDB:

By default, Terraform stores the state file (terraform.tfstate) locally. This is unsuitable for team collaboration as it can lead to conflicts and state corruption. The best practice is to store the state file remotely. I configured an S3 bucket to store the state file and a DynamoDB table for state locking.25

* **S3 Bucket:** Provides a durable, centralized location for the terraform.tfstate file. Versioning was enabled on the bucket as a safety measure to recover from accidental deletions or errors.
* **DynamoDB Table:** Provides a locking mechanism. When one team member runs terraform apply, a lock is placed in the DynamoDB table. If another user tries to run apply simultaneously, the command will fail, preventing concurrent modifications and potential state file corruption.25

Terraform Modules for Reusability:

As infrastructure grows, managing all resources in one file becomes unwieldy. Terraform modules allow for the encapsulation of related resources into a reusable, logical unit.27 I refactored the web server configuration (EC2 instance and security group) into a module.

*Module Structure:*

/modules  
 /web-server  
 main.tf # Defines aws\_instance and aws\_security\_group  
variables.tf # Defines input variables (e.g., instance\_type, ami\_id)  
 outputs.tf # Defines output values (e.g., public\_ip)

*Root main.tf consuming the module:*

Terraform

module "my\_web\_server" {  
 source = "./modules/web-server"  
instance\_type = "t2.micro"  
ami\_id = "ami-0c55b159cbfafe1f0"  
}  
  
output "web\_server\_ip" {  
 value = module.my\_web\_server.public\_ip  
}

Using modules promotes the Don't Repeat Yourself (DRY) principle, improves organization, and allows teams to share standardized infrastructure components.28

## 5.2. Project II Walkthrough: Configuring Web Servers with Ansible Playbooks

With the infrastructure provisioned by Terraform, this project focused on automating the configuration of the EC2 instance using Ansible. This demonstrates the separation of concerns: Terraform creates the "house," and Ansible furnishes it.

### 5.2.1. Ansible Setup and Inventory

1. **Control Node:** A designated EC2 instance (or a local machine) was set up as the Ansible control node where Ansible was installed.
2. **Managed Node:** The Project-WebServer instance created by Terraform was the managed node.
3. **SSH Connectivity:** Key-based SSH authentication was established from the control node to the managed node to enable Ansible's agentless communication.
4. **Inventory:** Initially, a static inventory file (/etc/ansible/hosts) was created to tell Ansible about the target host.

Ini, TOML  
[webservers]  
<web\_server\_public\_ip>ansible\_user=ec2-user ansible\_ssh\_private\_key\_file=~/.ssh/my-aws-key.pem

### 5.2.2. From Ad-hoc Commands to a Declarative Playbook

I first used Ansible's ad-hoc commands to perform individual tasks, which is useful for quick, one-off operations.

* **Ping Test:** ansible webservers -m ping (To verify connectivity)
* **Install Apache:** ansible webservers -m yum -a "name=httpd state=present" --become
* **Start Service:** ansible webservers -m service -a "name=httpd state=started enabled=yes" --become

These commands demonstrated **idempotency**. Running the installation command a second time resulted in an "ok" (green) status instead of "changed" (yellow), because Ansible detected that Apache was already installed.5

While useful, ad-hoc commands are not scalable. The next step was to codify this logic into an Ansible Playbook, a YAML file that defines a series of tasks.

**Web Server Configuration Playbook (configure-web.yml):**

YAML

---  
-name:ConfigureWebServer  
hosts:webservers  
become:true# Execute tasks with sudo privileges  
tasks:  
-name:1.InstallApachewebserver(httpd)  
yum:  
name:httpd  
state:present  
  
-name:2.Deploycustomindexpage  
copy:  
src:./index.html# A local file on the control node  
dest:/var/www/html/index.html  
  
-name:3.Ensurehttpdserviceisstartedandenabled  
service:  
name:httpd  
state:started  
enabled:yes

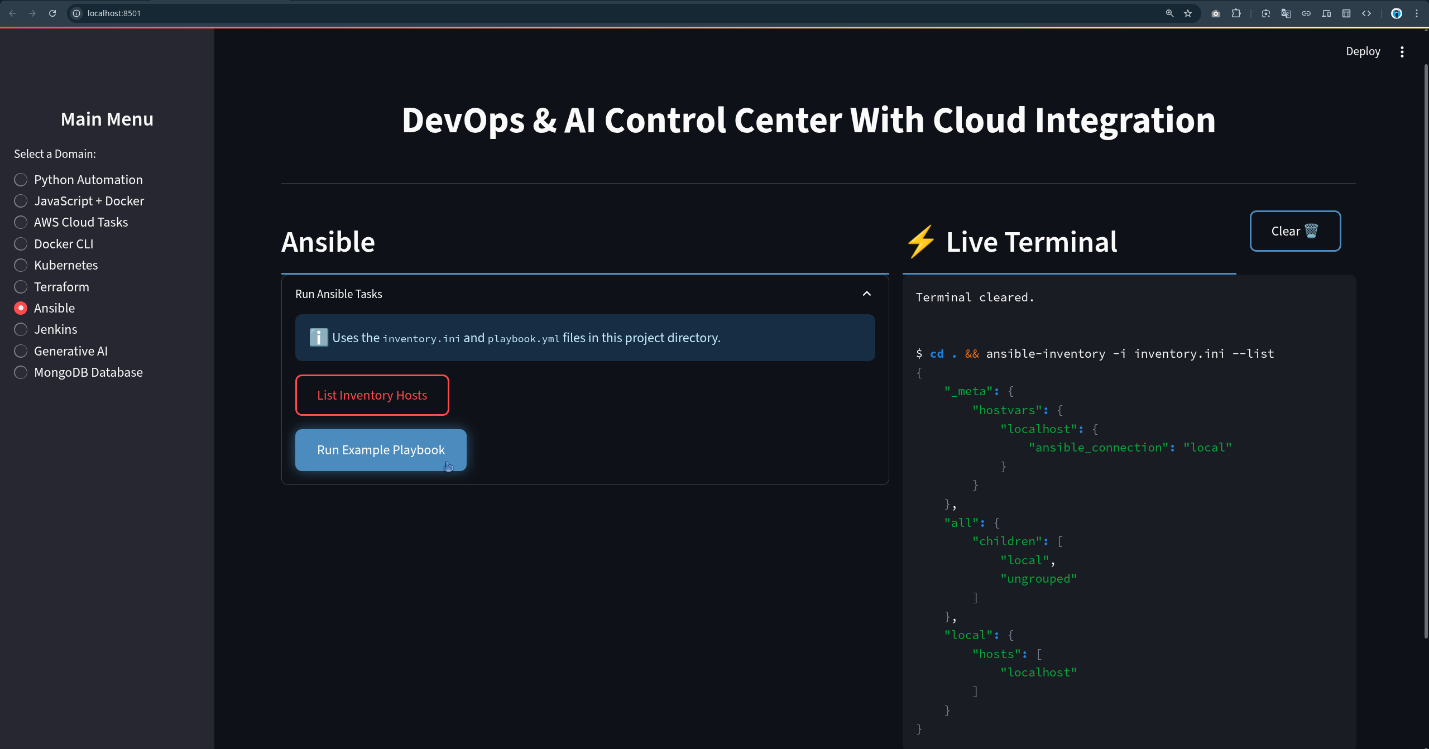


Figure 5. 3 Ansible

This playbook was executed using the command ansible-playbook configure-web.yml. It provides a declarative, self-documenting, and version-controlled way to define the desired state of the server.5

Table 5. 2: Common Ansible Modules Used in Web Server Configuration

|  |  |
| --- | --- |
| Module | Purpose |
| yum | Manages packages on Red Hat-based systems (e.g., install, remove). 5 |
| copy | Copy files from control nodes to managed nodes. 5 |
| service | Manages services on the managed node (e.g., start, stop, restart). 5 |
| file | Manages files, directories, and symlinks (e.g., create directories, set permissions). 5 |
| ping | A simple test module to check connectivity and Python interpreter is available. 5 |

### 5.2.3. Advanced Concepts for Scalable and Secure Automation

Ansible Roles for Better Organization:

To make the configuration more structured and reusable, the playbook was refactored into an Ansible Role. A role is a standardized directory structure for grouping tasks, variables, templates, and handlers.29

*Role Structure (apache-webserver):*

/roles  
 /apache-webserver  
 /tasks  
main.yml # Contains the tasks from the playbook  
 /handlers  
main.yml # Contains handlers, e.g., to restart httpd on config change  
 /files  
index.html # Static files to be copied

The main playbook then becomes much simpler, just calling the role:

YAML

---  
-name:ConfigureWebServerusingRole  
hosts:webservers  
become:true  
roles:  
-apache-webserver

Using roles is the standard for managing complex configurations, promoting modularity and sharing within teams.30

Ansible Vault for Securing Sensitive Data:

Playbooks often require sensitive information like passwords or API keys. Storing these in plaintext in a Git repository is a major security risk. Ansible Vault provides a mechanism to encrypt sensitive data within YAML files.31

I demonstrated this by encrypting a hypothetical variable:

1. Encrypt a variable:  
   ansible-vault encrypt\_string --ask-vault-pass 'mySecretPassword' --name 'db\_password'
2. **Use in playbook:** The encrypted output is pasted directly into a vars file.  
   YAML  
   db\_password:!vault|  
    $ANSIBLE\_VAULT;1.1;AES26  
    316339633...
3. Run the playbook: The vault password is provided at runtime.  
   ansible-playbook configure-web.yml --ask-vault-pass

This ensures that secrets are protected at rest within the version control system.32

AWS Dynamic Inventory:

Manually updating the static inventory file with IP addresses from Terraform is inefficient. Ansible can dynamically discover cloud resources using an inventory plugin. I configured the amazon.aws.aws\_ec2 inventory plugin.

1. **Configuration (aws\_ec2.yml):** A YAML file was created to configure the plugin, telling it to use AWS credentials and group hosts by their tags.

YAML  
# aws\_ec2.yml  
plugin:amazon.aws.aws\_ec2  
regions:  
-us-west-2  
keyed\_groups:  
-key:tags.Name# Group instances by the value of their 'Name' tag  
prefix:name

1. Execution: The playbook was then run using this dynamic inventory file.  
   ansible-playbook -i aws\_ec2.yml configure-web.yml

This setup creates a seamless workflow where Terraform provisions instances with specific tags, and Ansible automatically discovers and configures them without any manual intervention, a key pattern in mature cloud automation.33

## 5.3. Project III Walkthrough: Implementing ELB and Auto Scaling Groups

This project focused on transforming the single-server setup into a highly available, scalable, and fault-tolerant architecture using core AWS services. This is a critical step in preparing an application for a production workload. The entire architecture was defined using Terraform.

### 5.3.1. Architecture Design

The goal was to build an architecture that could automatically handle traffic spikes and recover from instance failures. The high-level design is as follows:

**(A figure would be included here showing the following flow)**

* An internet user makes a request.
* The request hits a Route 53 DNS record (not implemented in this project, but would be the next step).
* The DNS record points to an **Application Load Balancer (ALB)**.
* The ALB sits in public subnets across two or more **Availability Zones (AZs)** for high availability.
* The ALB distributes traffic to a **Target Group**.
* The Target Group contains EC2 instances managed by an **Auto Scaling Group (ASG)**.
* The ASG spans the same multiple AZs and automatically launches or terminates instances in private subnets based on defined scaling policies.
* A NAT Gateway in the public subnet allows instances in the private subnet to access the internet for updates, while remaining secure from inbound connections.

**Figure 5.1: High-Level Architecture for a Scalable Web Application on AWS**

### 5.3.2. Terraform Implementation

Application Load Balancer (ALB):

The ALB operates at Layer 7 (the application layer), making it ideal for HTTP/HTTPS traffic. It can make intelligent routing decisions based on paths, headers, or hostnames.5

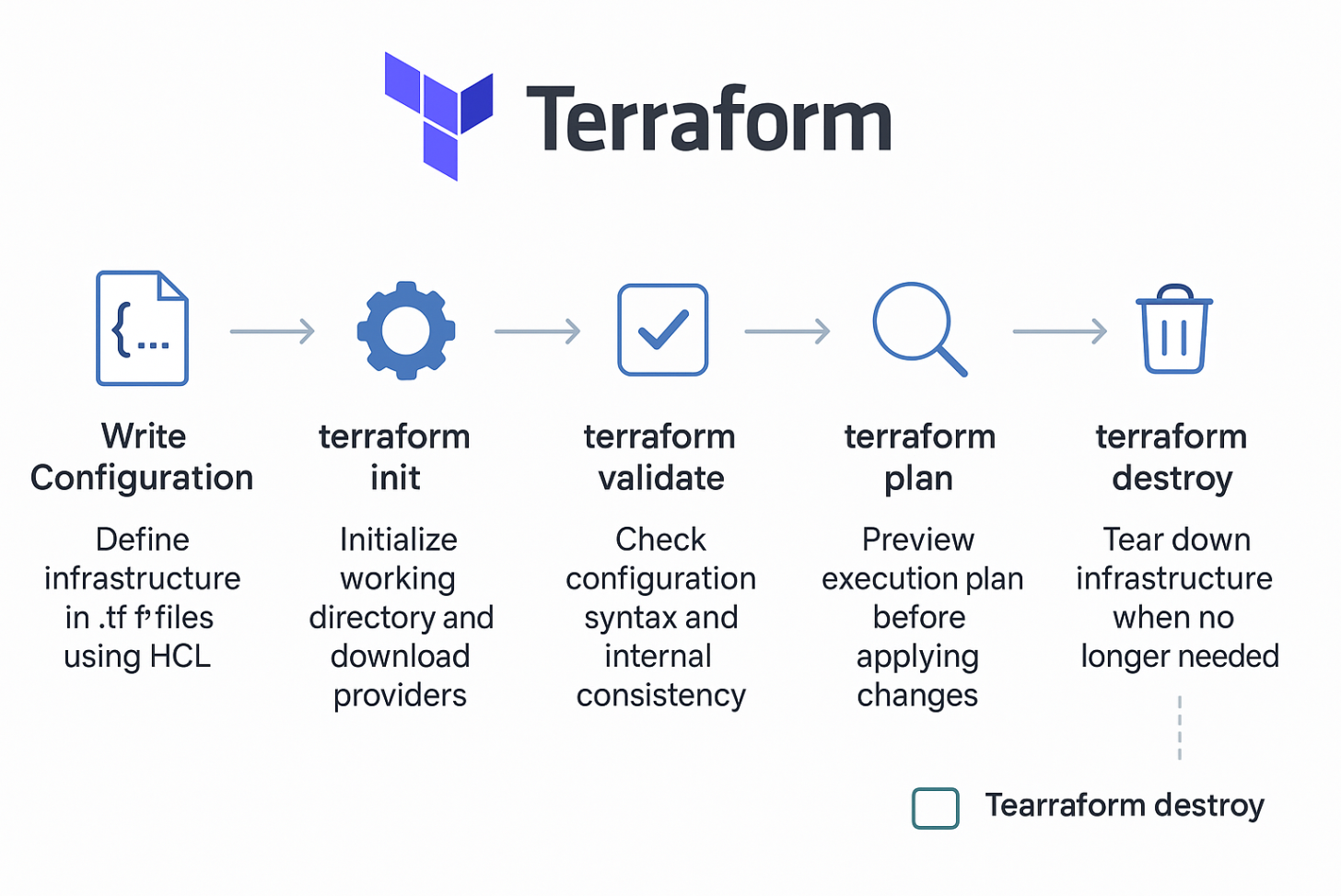


Figure 5. 4 Terraform Overview

Terraform

# alb.tf  
  
resource "aws\_lb" "main\_alb" {  
 name = "project-alb"  
 internal = false  
load\_balancer\_type = "application"  
security\_groups = [aws\_security\_group.alb\_sg.id] // A security group for the ALB  
 subnets = [aws\_subnet.public\_subnet\_a.id, aws\_subnet.public\_subnet\_b.id] // Span multiple AZs  
}  
  
resource "aws\_lb\_target\_group" "main\_tg" {  
 name = "project-tg"  
 port = 80  
 protocol = "HTTP"  
vpc\_id = aws\_vpc.main\_vpc.id  
health\_check {  
 path = "/index.html"  
 }  
}  
  
resource "aws\_lb\_listener" "http" {  
load\_balancer\_arn = aws\_lb.main\_alb.arn  
 port = "80"  
 protocol = "HTTP"  
  
default\_action {  
 type = "forward"  
target\_group\_arn = aws\_lb\_target\_group.main\_tg.arn  
 }  
}

This configuration creates the ALB, a target group to hold the backend instances, and a listener to forward traffic on port 80 to that target group. The health check ensures the ALB only sends traffic to healthy instances.5

Auto Scaling Group (ASG):

The ASG is the core component for scalability and fault tolerance. It ensures a specified number of instances are always running and can automatically adjust that number based on demand.5

1. **Launch Template:** First, a Launch Template was created. This acts as a blueprint for the instances the ASG will launch, specifying the AMI, instance type, key pair, security groups, and a user\_data script to install the web server upon launch.  
   Terraform  
   # asg.tf  
     
   resource "aws\_launch\_template" "web\_lt" {  
   name\_prefix = "web-server-lt-"  
   image\_id = "ami-0c55b159cbfafe1f0"  
   instance\_type = "t2.micro"  
   key\_name = "my-aws-key"  
   vpc\_security\_group\_ids = [aws\_security\_group.web\_sg.id]  
     
   user\_data = base64encode(<<-EOF  
    #!/bin/bash  
    yum update -y  
    yum install -y httpd  
   systemctl start httpd  
   systemctl enable httpd  
    echo "<h1>Deployed via ASG Launch Template</h1>" > /var/www/html/index.html  
    EOF  
    )  
   }
2. **Auto Scaling Group:** The ASG resource was then defined, linking to the Launch Template and the ALB's Target Group.

Terraform  
# asg.tf  
  
resource "aws\_autoscaling\_group" "main\_asg" {  
 name = "project-asg"  
launch\_template {  
 id = aws\_launch\_template.web\_lt.id  
 version = "$Latest"  
 }  
vpc\_zone\_identifier = [aws\_subnet.private\_subnet\_a.id, aws\_subnet.private\_subnet\_b.id]  
target\_group\_arns = [aws\_lb\_target\_group.main\_tg.arn]  
  
min\_size = 2 // Always keep at least 2 instances running  
max\_size = 5 // Scale out to a maximum of 5 instances  
desired\_capacity = 2 // Start with 2 instances  
}

### 5.3.3. Demonstration and Analysis

After running terraform apply, the following behavior was observed:

* The ASG automatically launched two EC2 instances, one in each specified private subnet, to meet the desired\_capacity.
* The ALB's target group registered these two new instances and, after they passed their health checks, marked them as "healthy."
* Accessing the ALB's public DNS name in a browser displayed the web page, with the load being distributed between the two instances.

To test fault tolerance, I manually terminated one of the EC2 instances. The ASG detected the unhealthy instance and automatically launched a new one to replace it, bringing the total back to the desired\_capacity of 2. This demonstrates the self-healing nature of the architecture.

To handle scalability, a **Target Tracking Scaling Policy** was added. This policy adjusts the number of instances to keep a specific metric (e.g., average CPU utilization) at a target value.

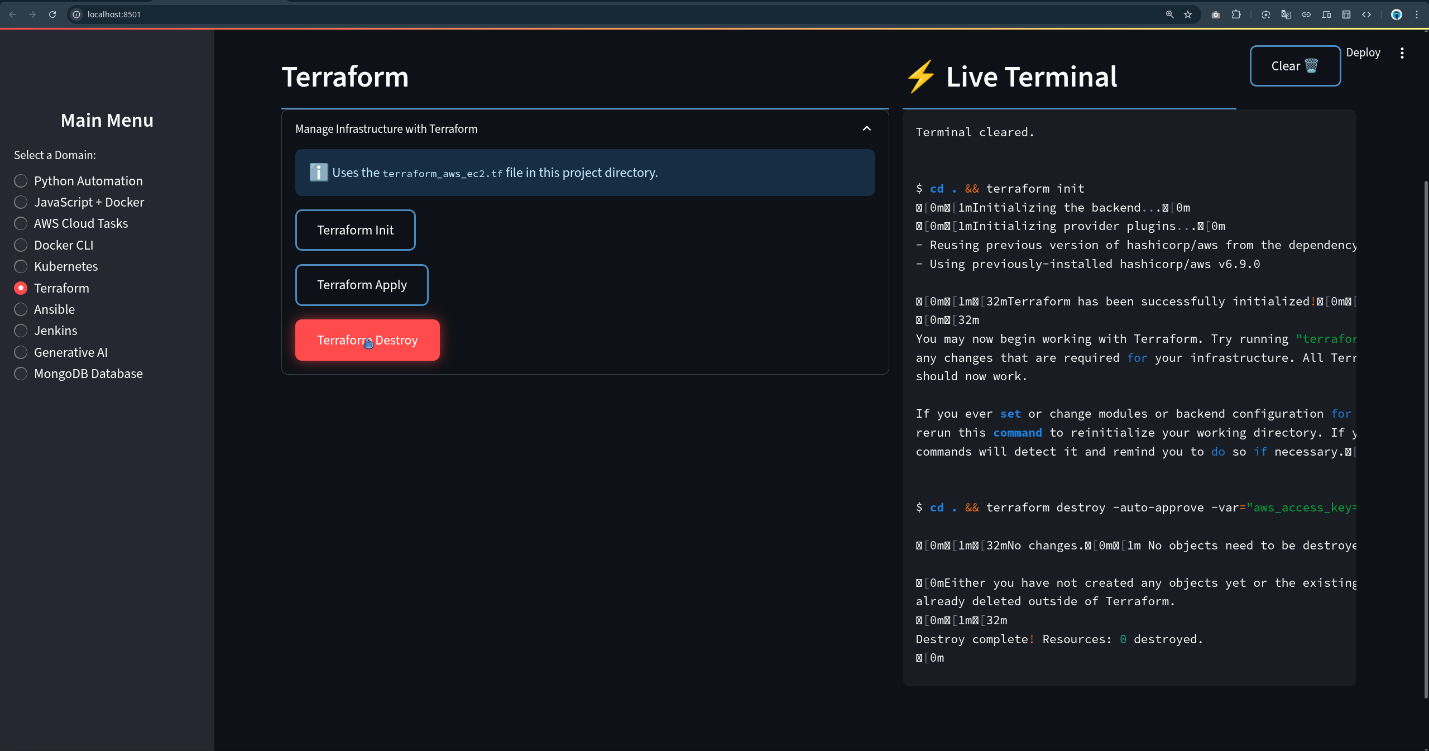


Figure 5. 5 Terraform

Terraform

resource "aws\_autoscaling\_policy" "cpu\_scaling\_policy" {  
 name = "cpu-target-tracking-policy"  
autoscaling\_group\_name = aws\_autoscaling\_group.main\_asg.name  
policy\_type = "TargetTrackingScaling"  
  
target\_tracking\_configuration {  
predefined\_metric\_specification {  
predefined\_metric\_type = "ASGAverageCPUUtilization"  
 }  
target\_value = 60.0 // Target 60% average CPU utilization  
 }  
}

With this policy, if a traffic spike caused the average CPU to exceed 60%, the ASG would automatically "scale out" by adding new instances. When traffic subsided, it would "scale in" by terminating unneeded instances, thus optimizing for both performance and cost.5 This project successfully demonstrated the core principles of designing modern, cloud-native applications that are resilient, scalable, and cost-effective.

## 5.4. Project IV Walkthrough: Building the Automated Jenkins CI/CD Pipeline

This capstone project integrated all the preceding concepts into a single, automated workflow. It involved taking a simple application from development to a deployable, containerized artifact, all orchestrated by Jenkins. This embodies the core value proposition of Continuous Integration and Continuous Deployment (CI/CD).

### 5.4.1. The Application and its Tests

* **Python Flask API:** A minimal web API was developed using the Python Flask framework. The application had a single endpoint /info that returned a JSON object with static company information. This served as the codebase for the pipeline.5

Python  
# app.py  
from flask import Flask, jsonify  
app = Flask(\_\_name\_\_)  
  
@app.route('/info')  
defget\_info():  
company\_data = {  
"companyName": "LinuxWorld",  
"services":  
 }  
returnjsonify(company\_data)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
app.run(host='0.0.0.0', port=5000)

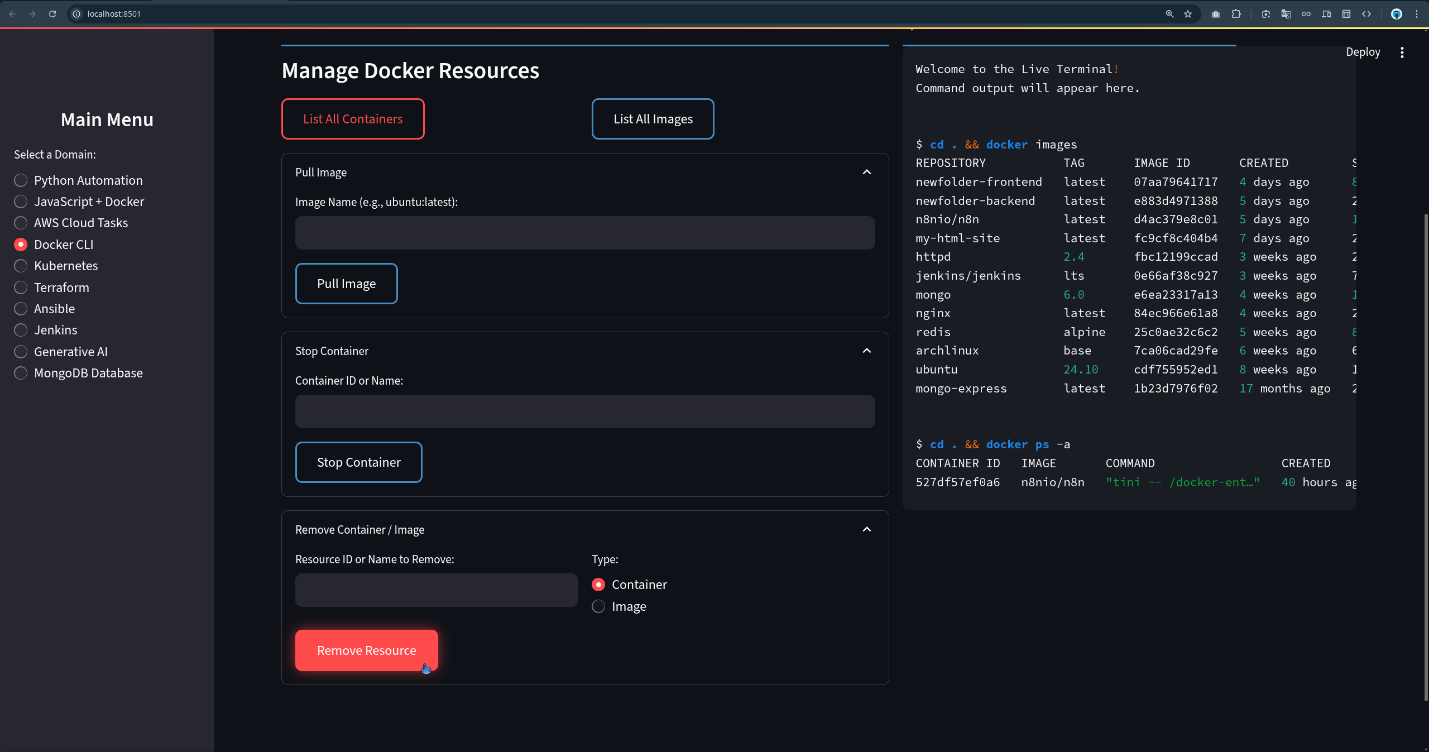
* **Automated Testing with Pytest:** To incorporate the "Test" phase of CI/CD, a simple unit test was written using the Pytest framework. This test verifies that the /info endpoint returns a successful status code (200) and that the response is valid JSON.5
* Python  
  # test\_app.py  
  from app import app  
  deftest\_info\_endpoint():  
   client = app.test\_client()  
   response = client.get('/info')  
  assertresponse.status\_code == 200  
  assertresponse.is\_json

### 5.4.2. Dockerizing the Application with Best Practices

To ensure environmental consistency and portability, the Flask application was containerized using Docker. The Dockerfile was created with a focus on optimization and security.

Multi-Stage Dockerfile:

A multi-stage build was implemented to create a small and secure final image. The first stage (builder) installs dependencies, and the second, final stage copies only the application code and dependencies into a clean, lightweight base image, discarding the build tools.35



**Figure 6.** 6 Docker Command Line Interface

Dockerfile

# Stage 1: The Builder Stage  
# Installs dependencies into a virtual environment  
FROM python:3.9-slim as builder  
WORKDIR /app  
COPY requirements.txt.  
RUN python -m venv /opt/venv  
ENV PATH="/opt/venv/bin:$PATH"  
RUN pip install --no-cache-dir -r requirements.txt  
  
# Stage 2: The Final Production Stage  
# Copies the application and the virtual environment from the builder  
FROM python:3.9-slim  
  
# Create a non-root user for security  
RUNadduser --disabled-password --gecos""appuser  
WORKDIR /app  
COPY --from=builder /opt/venv /opt/venv  
COPY app.py.  
COPY test\_app.py.  
  
# Set environment for the new user  
USERappuser  
ENV PATH="/opt/venv/bin:$PATH"  
  
# Command to run the application  
CMD ["python", "app.py"]

This approach significantly reduces the final image size and its attack surface by not including build-time tools in the production container.37 A

.dockerignore file was also used to exclude files like .git and \_\_pycache\_\_ from the build context, further optimizing the image.38

### 5.4.3. The Jenkins CI/CD Pipeline (Jenkinsfile)

Jenkins was installed on a dedicated EC2 instance with Docker and Git also installed. The Jenkins user was added to the docker group to allow it to execute Docker commands.39 The entire pipeline was defined as code in a

Jenkinsfile using the declarative pipeline syntax.

Table 5. 3: Jenkins Pipeline Stages and Their Functions

|  |  |  |
| --- | --- | --- |
| Stage | Purpose | Key Actions |
| **Checkout** | Retrieve the latest source code from the version control system. | git 'https://github.com/YashGupta/devops-project.git' |
| **Test** | Run automated tests to validate the code changes. | sh 'pytest' |
| **Build Image** | Build the Docker image from the Dockerfile. | docker.build("yashgupta/flask-api:${env.BUILD\_NUMBER}") |
| **Scan Image** | Perform security vulnerability scanning on the built image. | sh 'trivy image yashgupta/flask-api:${env.BUILD\_NUMBER}' |
| **Push Image** | Push the tagged, scanned image to a container registry (Docker Hub). | docker.withRegistry(...) { image.push() } |
| **Deploy** | (Conceptual) Update production environment with new image. | This would involve updating the ASG Launch Template and triggering an instance refresh. |

This pipeline was configured in Jenkins to trigger automatically via a GitHub webhook on every push event to the main branch. This setup ensures that every code change is automatically integrated, tested, secured, and packaged, making it ready for deployment at any time. This project successfully demonstrated the creation of a robust, automated, and secure software delivery lifecycle.

Advanced Concept: Jenkins Shared Libraries

For larger organizations with many pipelines, copying and pasting Groovy code is inefficient. Jenkins Shared Libraries allow for the creation of reusable pipeline code stored in a separate Git repository.40 I explored creating a custom step, for example,

dockerBuildAndPush, in a shared library. This would centralize the logic for building, scanning, and pushing images, allowing any pipeline to call it with a single line, thus enforcing consistency and simplifying maintenance.41

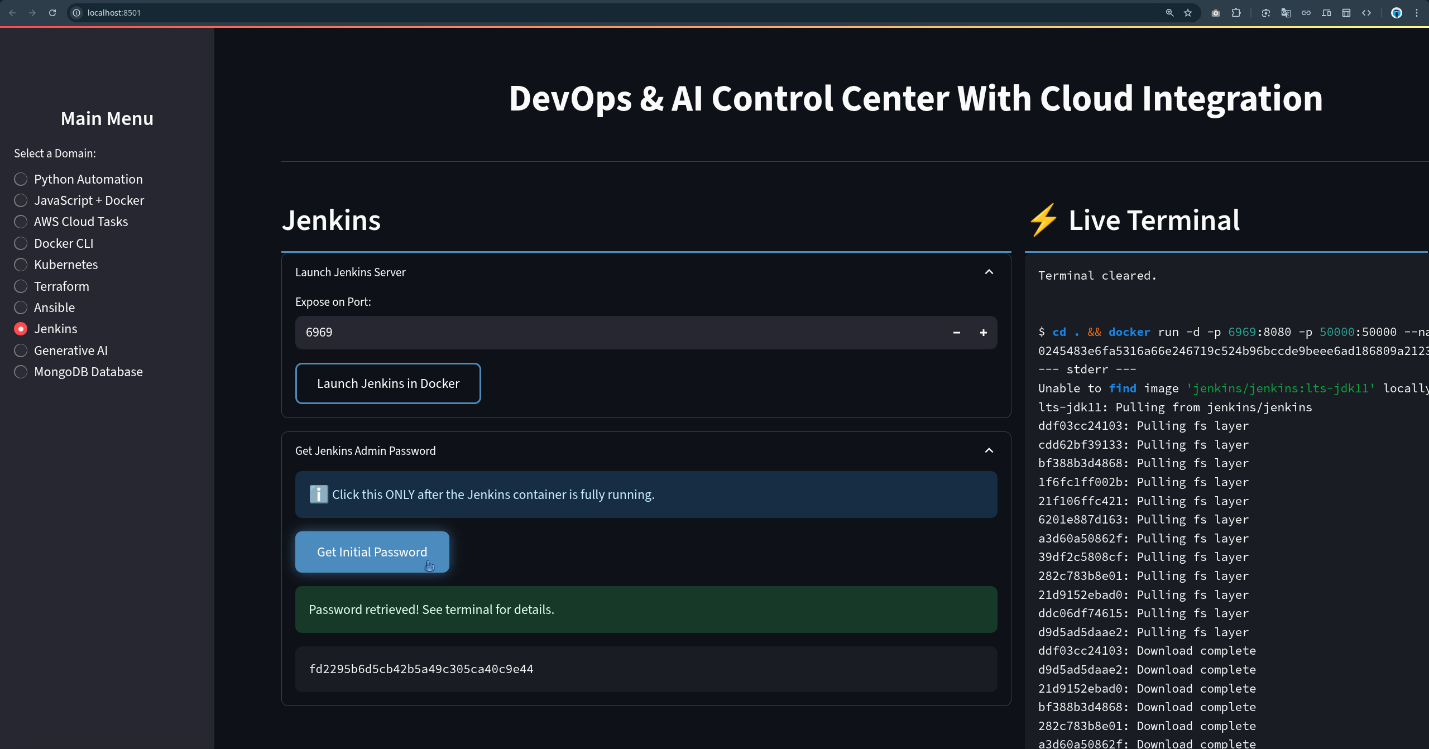


Figure 5. 7 Jenkins

## 5.5. Project V Analysis: A Practical Exploration of GenAI Ops

This final module was a research-oriented exploration into the cutting-edge intersection of Generative AI and DevOps, a field increasingly referred to as GenAI Ops. The objective was to understand and demonstrate how Large Language Models (LLMs) can function as "agents" to automate complex operational tasks, potentially transforming the role of the DevOps engineer.

### 5.5.1. Core Concepts: Agentic AI and Chain of Thought

* **Agentic AI:** This refers to a system where an AI model, typically an LLM, is not just a passive text generator but an active agent capable of reasoning, planning, and executing tasks. It achieves this by interacting with a set of external "tools," which could be anything from a Linux shell and cloud provider APIs to a web search or a database client.5 The LLM acts as the "brain," determining which tool to use and with what parameters to fulfill a user's request given in natural language.
* **Chain of Thought (CoT) Reasoning:** This is the process that enables Agentic AI. When given a complex prompt, the LLM breaks the problem down into a series of intermediate, logical steps, much like human reasoning. It "thinks" about the sequence of commands or actions needed before executing them. This makes its problem-solving process more transparent and often more accurate than a direct, monolithic response.5

This paradigm represents a significant evolution in automation. Instead of an engineer needing to know the exact syntax of a Terraform resource or an Ansible module, they can describe their desired outcome in plain English. The AI agent, using its reasoning capabilities, translates this intent into executable code.

### 5.5.2. Practical Demonstration in a Jupyter Notebook

To make this concept tangible, I used a Jupyter Notebook environment to interact with Google's Gemini LLM, a scenario reflective of the hands-on labs during the internship.5 The goal was to have the AI agent generate a functional Ansible Playbook from a natural language prompt.

**Prompt:**

"You are a DevOps expert specializing in Ansible. Write a complete Ansible Playbook that performs the following actions on a group of hosts named [webservers]:

1. Install the nginx web server.
2. Create a custom index.html file in the default Nginx document root (/usr/share/nginx/html) with the content 'Welcome to the AI-configured server!'.
3. Ensure the nginx service is started and enabled to run on boot."

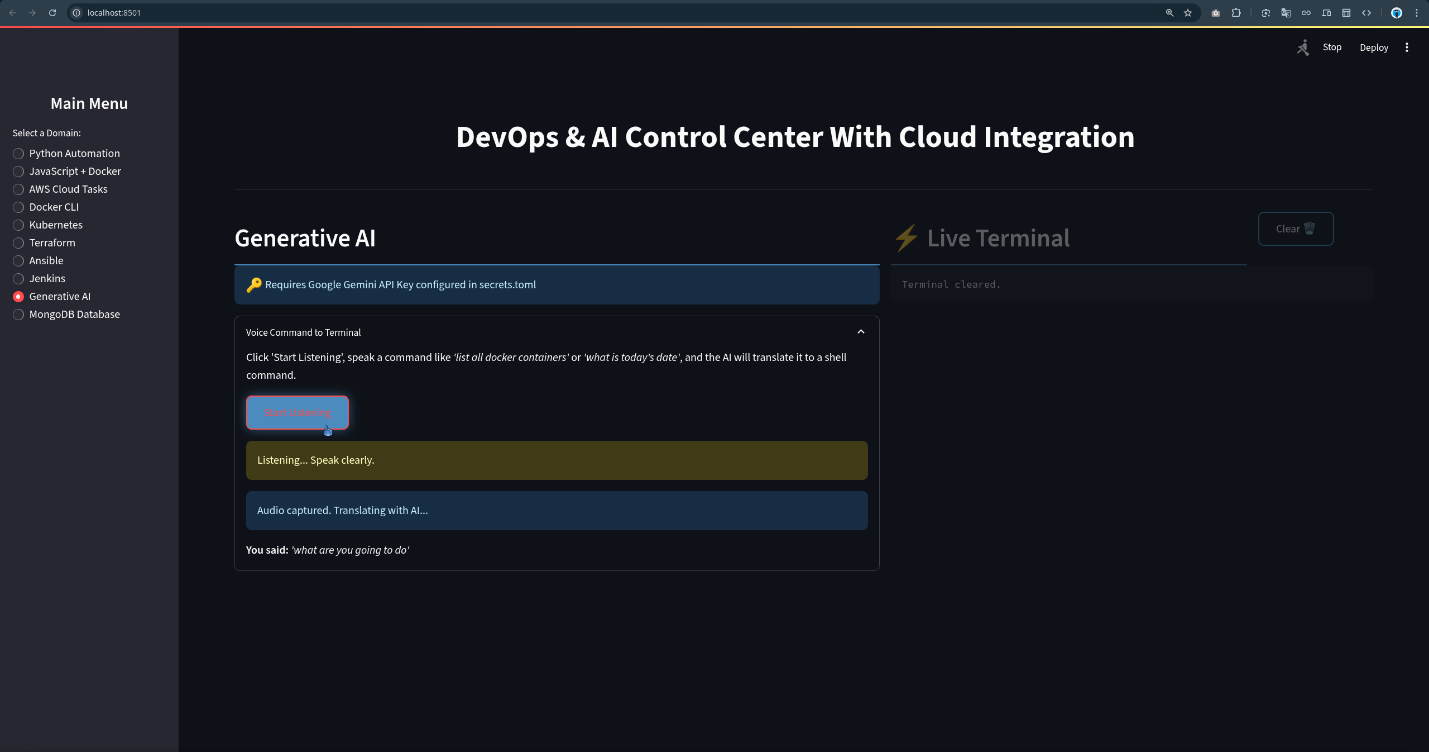


Figure 5. 8 Generative Artificial Intelligence

**LLM-Generated Output (Analyzed):**

YAML

---  
-name:ConfigureNginxWebServer  
hosts:webservers  
become:true  
tasks:  
-name:InstallNginx  
ansible.builtin.package:  
name:nginx  
state:present  
  
-name:Createcustomindex.html  
ansible.builtin.copy:  
content:"<h1>Welcome to the AI-configured server!</h1>"  
dest:/usr/share/nginx/html/index.html  
owner:root  
group:root  
mode:'0644'  
  
-name:StartandenableNginxservice  
ansible.builtin.service:  
name:nginx  
state:started  
enabled:true

**Analysis of the Generated Code:**

* **Correctness:** The generated playbook is syntactically correct and logically sound. It correctly uses the package, copy, and service modules.
* **Best Practices:** The agent demonstrated good practices by using fully qualified collection names (e.g., ansible.builtin.package), which is the modern standard. It also correctly set file permissions (mode: '0644') on the copied file, a detail a novice might overlook.
* **Limitations:** The code is generic. It assumes a package manager that ansible.builtin.package can handle (like yum or apt). It doesn't include handlers for restarting the service on configuration change, which is a more advanced Ansible practice.

### 5.5.3. The Future Role of the DevOps Engineer

This practical experiment provided a clear window into the future of DevOps. The rise of Agentic AI is unlikely to eliminate the need for DevOps engineers. Instead, it will fundamentally transform the role, shifting the required skills from direct implementation to strategic oversight.

The traditional role requires deep, explicit knowledge of the syntax and semantics of numerous tools. An engineer must know precisely how to write a Terraform resource or an Ansible task. The future role, augmented by AI, will pivot towards a different set of competencies:

1. **Expert Prompt Engineering:** The ability to craft clear, unambiguous, and context-rich prompts to guide the AI agent towards the desired outcome will be paramount.
2. **Critical Validation and Security Review:** The AI's output, while often correct, cannot be trusted blindly. The engineer's role will be to act as an expert reviewer, validating the generated code for correctness, efficiency, and, most importantly, security. The AI has no intrinsic understanding of an organization's specific security policies or compliance requirements.
3. **Systems Architecture and Integration:** The engineer will focus more on the high-level architecture—how the AI-generated components fit together into a larger, cohesive, and secure system—rather than on the low-level code for each component.

The foundational knowledge gained throughout this internship—understanding *how* Terraform provisions resources, *how* Ansible enforces state, and *how* a CI/CD pipeline functions—is the essential prerequisite for this future role. Without this deep understanding, an engineer cannot effectively prompt, validate, or orchestrate the AI agents that will increasingly automate the tasks of tomorrow.

# Chapter 6

# Training Outcome

This internship at LinuxWorld has been a transformative experience, providing not just theoretical knowledge but, more importantly, extensive hands-on practice with the tools and methodologies that define modern IT operations. This chapter summarizes the tangible outcomes of this training, including the technical competencies I have acquired, the practical application of DevOps principles, the problem-solving skills I have developed, and my overall professional growth.

## 6.1. Summary of Acquired Technical Competencies

The project-based learning approach enabled me to develop a comprehensive and interconnected skill set across the entire DevOps lifecycle. My key technical competencies now include:

* **Cloud Infrastructure Management (AWS):** Proficient in designing, deploying, and managing core AWS services, including VPC for networking, EC2 for compute, S3 and EBS for storage, and IAM for security and access control.
* **Infrastructure as Code (IaC):** Mastered the use of Terraform to automate the provisioning and management of cloud infrastructure, including advanced concepts like remote state management, modular design, and workspace utilization for multi-environment setups.
* **Configuration Management:** Gained expertise in using Ansible to automate software installation and configuration on servers. This includes writing declarative playbooks, organizing configurations into reusable roles, securing secrets with Ansible Vault, and integrating with cloud environments via dynamic inventories.
* **Containerization:** Developed a strong capability in using Docker to package applications into portable containers. This includes writing optimized, multi-stage Dockerfiles and understanding best practices for creating lightweight and secure images.
* **CI/CD Pipeline Orchestration:** Acquired the ability to design, implement, and manage end-to-end CI/CD pipelines using Jenkins. This involves integrating with source control (Git/GitHub), automating testing (Pytest), building and scanning Docker images, and orchestrating the flow from code commit to a deployable artifact.
* **Scripting and Automation:** Enhanced my proficiency in Python for application development (using the Flask framework) and automation, and Bash scripting for essential Linux system administration and pipeline tasks.
* **Monitoring and Observability:** Gained a foundational understanding of monitoring principles using AWS CloudWatch and an introduction to advanced open-source tools like Prometheus and Grafana for creating sophisticated monitoring dashboards.
* **Version Control:** Solidified my skills in using Git and GitHub for collaborative development, versioning all artifacts as code, from the application itself to the infrastructure and pipeline definitions.

## 6.2. Application of DevOps Principles in Practice

Beyond learning individual tools, the most significant outcome was understanding how to apply the core principles of DevOps in a practical context. The projects were a direct implementation of the CALMS framework and The Three Ways:

* **Flow and Automation:** Project I (Terraform) and Project II (Ansible) directly addressed the First Way by automating the entire workflow from bare infrastructure to a configured server. This eliminated manual bottlenecks and dramatically increased the flow and speed of environment creation.
* **Fast Feedback:** Project IV (CI/CD Pipeline) was a powerful implementation of the Second Way. By automatically running tests and security scans on every code commit, the pipeline provided immediate feedback to the developer, allowing for the early detection and correction of bugs and vulnerabilities.
* **Continuous Learning and Experimentation:** The modular nature of the projects, especially the use of Terraform modules and Ansible roles, created a safe environment for experimentation. New configurations could be tested in isolated workspaces (Project I) or on new branches, fostering the Third Way's culture of continuous learning without risking the stability of the main environment.
* **Everything as Code:** The entire project, from the AWS infrastructure (.tf files) and server setup (.yml playbooks) to the delivery pipeline (Jenkinsfile), was defined as code and stored in Git. This embodied the foundational DevOps tenet of "Everything as Code," making the entire system transparent, versionable, and auditable.

## 6.3. Challenges Encountered and Problem-Solving Strategies

The internship was not without its technical hurdles. Overcoming these challenges was one of the most valuable aspects of the learning process, as it honed my problem-solving and troubleshooting skills.

* **Challenge 1: IAM Permission Errors:** When first integrating Terraform and Ansible with AWS, I frequently encountered AccessDenied errors. It was initially difficult to pinpoint the exact missing permission required for a specific API call.
  + **Solution:** I learned to systematically debug these issues by using AWS CloudTrail to inspect the failed API calls and identify the specific action that was denied. This allowed me to craft more precise, least-privilege IAM policies instead of resorting to overly broad permissions, which is a critical security practice.
* **Challenge 2: Complex VPC Networking:** Configuring the VPC with public and private subnets, route tables, an Internet Gateway, and a NAT Gateway was complex. Initially, instances in the private subnet could not access the internet to download packages.
  + **Solution:** I methodically traced the network path, verifying the route tables associated with the private subnet, the configuration of the NAT Gateway in the public subnet, and the security group rules. This process taught me the importance of a deep, fundamental understanding of cloud networking principles.
* **Challenge 3: Idempotency Issues in Ansible:** An early version of an Ansible playbook used shell commands instead of dedicated modules. This caused the playbook to re-run tasks unnecessarily and was not idempotent.
  + **Solution:** I refactored the playbook to use Ansible's built-in modules (yum, copy, service). This not only made the playbook idempotent but also more readable and platform-agnostic. I learned to consult the Ansible documentation (ansible-doc) frequently to find the appropriate module for a given task.
* **Challenge 4: Jenkins and Docker Integration:** Getting Jenkins to correctly build Docker images and communicate with the Docker daemon required careful configuration, especially regarding user permissions on the Linux host.
  + **Solution:** Troubleshooting involved checking the Jenkins console output for detailed error messages, verifying that the jenkins user was part of the docker group, and restarting the Jenkins service to apply the new group membership. This reinforced the importance of understanding the underlying Linux operating system on which the DevOps tools run.

## 6.4. Personal and Professional Growth

This internship has been instrumental in my development as a budding technology professional. On a personal level, it has instilled in me a greater sense of confidence in my ability to tackle complex technical problems. The process of learning, implementing, failing, and succeeding has been incredibly rewarding.

Professionally, this experience has prepared me for a career as a DevOps Engineer. I have moved beyond a theoretical understanding of tools to a practical appreciation of how they integrate to solve real business problems. I have learned the importance of automation, not for its own sake, but for the business value it delivers in terms of speed, reliability, and security. The emphasis on best practices—like using remote state, writing modular code, securing secrets, and building optimized images—has provided me with a mindset geared towards building production-ready systems. I leave this internship not just with a list of skills, but with a holistic understanding of the DevOps culture and its transformative impact on the technology industry.

# Chapter 7

# Conclusion

This internship at LinuxWorld has provided a comprehensive and deeply practical education in the principles and practices of DevOps with Cloud Computing. The journey, from understanding foundational concepts to implementing a fully automated, multi-faceted project, has culminated in a robust skill set directly aligned with the most pressing needs of the modern IT industry. This conclusion synthesizes the key achievements of the project, reflects on the broader business impact of the implemented strategies, and looks forward to potential future enhancements.

## 7.1. Synthesis of Project Achievements

The series of interconnected projects successfully demonstrated an end-to-end understanding of the modern software delivery lifecycle. The key achievements represent a mastery of the core competencies required of a modern DevOps professional:

* **Codified and Automated Infrastructure:** I successfully transitioned from manual, error-prone infrastructure management to a fully automated, code-based approach using Terraform. This achievement is significant as it establishes a single source of truth for the cloud environment, enabling the ability to provision a complete, secure, and production-ready AWS architecture in a repeatable and version-controlled manner, drastically reducing setup time and ensuring consistency across all deployments.5
* **Declarative and Consistent Configuration:** Using Ansible, I demonstrated the ability to declaratively enforce a desired state on servers. This is a critical achievement because it effectively eliminates "configuration drift"—the common problem where servers in an environment become inconsistent over time. The idempotent nature of the playbooks ensures that all environments are configured reliably, thereby reducing the potential for human error and simplifying the process of scaling out new servers.5
* **Resilient and Scalable Architecture:** I successfully architected and implemented a cloud-native application environment using AWS Load Balancers and Auto Scaling Groups. The resulting system is not just robust but also intelligent; it is fault-tolerant, capable of self-healing from instance failures, and can dynamically scale to meet fluctuating demand. This ensures high availability for users and optimizes operational costs by aligning resource consumption with real-time traffic.5
* **Fully Automated CI/CD Pipeline:** The capstone project culminated in a functional CI/CD pipeline orchestrated by Jenkins. This pipeline automates the entire software development lifecycle, from a developer's code commit to the creation of a tested, secured, and containerized application artifact ready for deployment. This achievement embodies the core DevOps goal of enabling rapid, reliable, and frequent software releases, forming the backbone of an agile development culture.5
* **Engagement with Emerging Technology:** The research and practical application of Agentic AI concepts demonstrated a proactive engagement with the future of automation. This showcases an ability to look beyond current tools and understand the long-term trajectory of the industry, where natural language prompts can be used to generate complex configurations, positioning me to adapt to the next wave of GenAI Ops.5

## 7.2. The Business Impact of an Integrated DevOps and Cloud Strategy

The technical achievements of this project are not merely academic exercises; they translate directly into tangible and significant business value. The integrated DevOps and Cloud strategy implemented here addresses key challenges faced by modern enterprises and provides a clear competitive advantage:

* **Accelerated Time-to-Market:** By automating the provisioning and deployment processes, the time required to move a new feature from an idea to production is drastically reduced from weeks or months to mere hours or minutes. This agility allows a business to respond more quickly to market changes, outpace competitors, and deliver value to customers faster.17
* **Improved Reliability and Availability:** The use of Infrastructure as Code ensures consistency and reduces the human error inherent in manual processes. The high-availability architecture, with its automated load balancing and self-healing auto-scaling, minimizes downtime. This leads to a more stable and reliable service, which directly enhances customer experience, builds trust, and protects revenue streams that depend on application availability.5
* **Enhanced Security Posture (DevSecOps):** By integrating security practices directly into the automated pipeline—a concept known as "Shift Left" or DevSecOps—potential vulnerabilities are identified and mitigated much earlier in the development cycle. Practices such as automated vulnerability scanning of Docker images and securing secrets with Ansible Vault reduce the cost and impact of security breaches, ensuring a more robust and compliant security posture from the ground up.43
* **Increased Operational Efficiency and Cost Optimization:** Automating repetitive manual tasks frees up highly skilled engineering teams from operational toil, allowing them to focus on innovation and value-added work. Furthermore, the pay-as-you-go nature of the cloud, combined with auto-scaling, ensures that infrastructure costs are dynamically optimized to match actual usage, preventing over-provisioning and reducing unnecessary expenditure.5

Ultimately, the strategies and tools learned and implemented during this internship enable a business to deliver higher-quality software, faster and more securely. This is the cornerstone of competitive advantage in today's digital-first economy.

## 7.3. Future Scope and Recommended Enhancements

While the projects successfully demonstrated a complete and functional workflow, they also serve as a strong foundation for further enhancements and more advanced, enterprise-grade implementations. The following represent logical next steps to further mature the system and align it with cutting-edge industry practices:

* **Transition to Container Orchestration with Kubernetes:** The current deployment model involves running a Docker container on EC2 instances managed by an Auto Scaling Group. The next evolutionary step would be to deploy the application to a managed Kubernetes cluster, like Amazon EKS. This would enable true microservices management, providing more sophisticated service discovery, declarative scaling, rolling updates, and advanced networking capabilities at the container level, which is the industry standard for complex, distributed applications.5
* **Implement Advanced Observability with Prometheus and Grafana:** While AWS CloudWatch provides excellent foundational monitoring for infrastructure, integrating Prometheus and Grafana would allow for more advanced, custom observability. This moves beyond simple monitoring ("what is broken?") to observability ("why is it broken?"). Prometheus could be used to scrape detailed, application-level metrics directly from the Flask API, and Grafana could be used to build rich, interactive dashboards that correlate system, application, and business metrics in a single, unified view.5
* **Strengthen DevSecOps Practices:** The security posture could be significantly enhanced by integrating more automated security tools into the CI/CD pipeline. This could include Static Application Security Testing (SAST) tools to scan the Python source code for vulnerabilities before the build stage, and Dynamic Application Security Testing (DAST) tools to probe the running application in a staging environment. Furthermore, implementing more granular, role-based IAM policies that strictly adhere to the principle of least privilege would further harden the cloud environment against unauthorized access.5
* **Explore Multi-Cloud and Hybrid Cloud Strategies:** The use of vendor-neutral tools like Terraform and Ansible provides a strong foundation for a multi-cloud strategy. A future project could involve adapting the Terraform and Ansible code to deploy the same application architecture on a different cloud provider, such as Microsoft Azure or Google Cloud Platform. This would demonstrate the flexibility and strategic advantage of avoiding vendor lock-in, enabling the business to leverage best-of-breed services from multiple clouds and meet complex data residency or disaster recovery requirements.5

# References

* Amazon Web Services. (2024). *EC2 Auto Scaling Documentation*.
* Ansible. (2024). *Ansible Documentation*.
* Atlassian. (2023). *The Evolution of IT Infrastructure Management*.
* Docker Inc. (2025). *Docker and Jenkins: Build Robust CI/CD Pipelines*.
* Gartner, Inc. (2025). *Reports on Public Cloud Services Market in India*.
* HashiCorp. (2024). *Terraform Documentation*.
* Jenkins Project. (2024). *Jenkins User Documentation*.
* LinuxWorld Informatics Pvt. Ltd. (2024). *Company Information and Training Programs*.
* NASSCOM. (2022). *Reports on Cloud Computing and Skills in India*.
* Red Hat, Inc. (2023). *Information on Ansible Automation Platform and Partnerships*.
* Internal Internship Documents: intern\_report.txt, page\_report.txt, topics\_intern.txt, timeline\_intern.txt.