Acknowledgement

I am indebted to our Principal **Dr. S.A.M.N Quadri** and management of K.C.T Engineering College for providing an environment with all facilities that helped me in completing the internship.

I am extremely grateful to **Dr. Sabera Begum**, HOD of the Computer Science and Engineering Department for her moral support and encouragement.

I wish to express my sincere gratitude to my internship coordinators **Prof. Sania Fatima** and **Prof. Mirza Arif Baig** from the Computer Science & Engineering Department, for their guidance and suggestions.

I thank all the teaching and non-teaching staff of the Department of Computer Science and Engineering for their kind help.

I also thank **Visvesvaraya Technological University, Belagavi** for providing me with this golden opportunity to do the internship at **Rooman Technologies** on the topic **AI - DevOps Engineer** which helped me in learning new things.

Last but not the least, I would like to add some personal notes. If there is a driving force that keeps me going, and what has not changed, it is the constant support and blessing of my parents, family and friends. There is no doubt, in spite of my strenuous efforts; error might remain in the internship report. Naturally, I take full responsibility for any lack of clarity, occasional erratum or inexactness that may occur.

MOHAMMED SAMIUDDIN
(3KC21CS030)

ABSTRACT

The AI - DevOps Engineer internship at Rooman Technologies, Bengaluru, provided a rigorous foundation in modern DevOps practices, tools, and methodologies to enhance software development and deployment workflows. The training encompassed Python programming (basic and advanced), Linux system administration, networking, database management, and cloud computing (AWS, GCP). Hands-on assignments included configuring CI/CD pipelines with Jenkins, containerizing applications using Docker, orchestrating deployments with Kubernetes, and automating infrastructure with Terraform. A key project involved developing a Multi-Cloud To-Do Application, deployed across AWS and IBM Servieces using Docker and Kubernetes. The implementation emphasized high availability, scalability, and fault tolerance by leveraging AWS EC2, S3, EKS, and GCP GKE. Additional explorations included AI-powered coding assistants like GitHub Copilot and practical applications of prompt engineering for efficient development. The internship reinforced critical DevOps principles such as automation, continuous integration, and Agile methodologies while enhancing technical proficiency in cloud-native technologies. The experience highlighted the significance of adaptability and lifelong learning in the evolving DevOps landscape, equipping participants with industry-relevant skills for scalable and resilient software delivery.

Contents

List of Figures

CHAPTER 1

COMPANY PROFILES

1.1 Profile - I

Name	Rooman Technologies Pvt Ltd
Address	#30, 12th Main, 1st Stage Rajajinagar, Bengaluru, Karnataka - 560 010
Contact Number	+91 8069451000
Email	online@rooman.net
Website	https://www.rooman.net
Company Registration Number	U72900KA1999PTC025311
Type of the Company	Private
Nature of the Company	IT-ITeS (Information Technology)
Company Logo	
Vision	Impart quality training for empowerment of youth to make India skill capital of the world. Integrate Global Technologies to introduce innovative Products and Solutions. Increase global presence through Associations, Collaborations and Partnerships. To setup Global Education Campus housing top universities of the world.
Company Operational Status	Private

1.2 Profile - II

Name	IBM India Private Limited
Address	No. 12, Subramanya Arcade, Bannerghatta Main Road, Bengaluru, Karnataka – 560029
Contact Number	+91 8040114047
Email	rccindia@in.ibm.com
Website	https://www.ibm.com/in-en
Company Registration Number	U72200KA1997PTC022382
Type of the Company	Private
Nature of the Company	IT-ITeS (Information Technology)
Company Logo	IBM
Vision	to equip students with job-ready skills in cutting-edge technologies like AI, cloud computing, and cybersecurity, thereby preparing them for successful careers. IBM seeks to encourage a culture where new ideas and different viewpoints can contribute to innovative solutions.
Company Operational Status	Private

1.3 Profile - III

Name	Wadhwani Foundation
Address	6th Floor, Bhive Premium, 48, Church Street, Bengaluru – 560 001
Contact Number	+91 8046196000
Email	marketing@wfglobal.org
Website	https://wadhwanifoundation.org/
Company Registration Number	U74999DL2018NPL333983
Type of the Company	Not-for-profit organization
Nature of the Company	Entrepreneurship, Skilling, Innovation and Research
Company Logo	WADHWANI FOUNDATION
Vision	To accelerate job growth in emerging economies and enable millions to earn a family-sustaining wage and lead a dignified life.
Company Operational Status	Private

CHAPTER 2

ABOUT THE COMPANY

Rooman Technologies Private Limited was founded in 1999 by a group of technology enthusiasts with the vision of establishing a training center of excellence in Bangalore, India.

2.1 History

Rooman Technologies Private Limited was founded in 1999 by a group of passionate technology enthusiasts with the primary goal of establishing a premier training center in Bangalore, India. Initially focused on creating a ”center of excellence,” the company prioritized recruiting top-tier trainers and fostering a vision to deliver training that ensures employability, instills confidence, and provides valuable certifications. The founding team and board members diligently tracked global and Indian technological advancements, leading to the development of innovative training concepts designed for long-term relevance. From its inception, Rooman Technologies has been committed to enhancing skills and employability, evident in its early work with the government on upskilling and reskilling programs. Today, Rooman Technologies stands as a testament to its initial vision, boasting a significant presence across India with government-supported training and vocational courses aimed at empowering graduates and students for roles as technicians, operators, and support assistants.

2.2 Achievements in Business

Rooman Technologies Private Limited has carved a distinguished path in the IT training sector over the past 24 years, marked by a deep commitment to ”Transforming Lives Through Training and Technology.” Key achievements that underscore their impact include:

- **Extensive Reach and Impact:** Successfully trained over 1.2 million students in IT-related fields, a number that continues to grow.
- **Government Recognition and Collaboration:** Proudly associated with the Government of India in its mission to establish India as the Skill Capital of the world, actively contributing to national upskilling initiatives.
- **Pioneering Training Models:** Raised the bar in online education by introducing the innovative FLIPPED CLASS MODEL, enhancing engagement and learning outcomes.

- **Omnichannel Presence:** Established itself as the only omnichannel training institute in India, boasting a robust physical presence with over 100 training centers nationwide, complementing its online offerings.
- **Comprehensive Training Portfolio:** Meticulously built a wide array of training services and products in critical areas like networking, software, and hardware, catering to current industry demands.
- **Industry-Recognized Certifications:** Provides certification courses for leading technology vendors such as Cisco, Microsoft, and Redhat, enhancing the credibility and value of their training.
- **Diversified Technological Engagement:** Expanded beyond training to develop products in the burgeoning IoT and ERP domains, alongside offering vital services in cybersecurity, data security, and data center support.
- **Focus on Trainer Excellence:** Implemented best-in-class ”train-the-trainer” methods to ensure their educators are highly motivated and updated with the latest technological advancements.
- **State-of-the-Art Infrastructure:** Evolved and reinvented their classroom infrastructure to provide an optimal learning environment with integrated study rooms, discussion lounges, knowledge banks, and specialized labs for hardware, networking, and software across all their centers.

2.3 Overall Turnover

With 24 years of experience, a pan-India network of over 100 training centers, and a track record of training over 1.2 million students, Rooman Technologies Private Limited demonstrates a significant scale of operations. Their engagement in diverse training programs, including government-backed initiatives and industry-recognized certifications, coupled with their expansion into product development and IT services, suggests a substantial overall turnover. Their position as a leading omnichannel training provider further contributes to their revenue streams.

2.4 About the Department (Training and Technology)

At the heart of Rooman Technologies Private Limited lies its Training and Technology department, driven by the core principle that ”Teaching and Technology is at the heart of each Rooman-ite.” This department is dedicated to the creation and delivery of high-quality training programs across networking, software, and hardware domains. They are committed to adopting the best ”train-the-trainer” methodologies to ensure their instructors remain motivated and abreast of the latest technological trends. The department continuously evolves its classroom infrastructure, providing students with best-in-class learning environments that foster collaboration, including well-equipped study rooms, discussion lounges, knowledge banks, and specialized labs for hardware, networking, and software. Furthermore, this department extends its expertise beyond traditional training to develop innovative products in the IoT and ERP sectors and provide crucial IT services like cybersecurity, data security, and data center support, demonstrating a holistic approach to leveraging technology for both education and practical application.

CHAPTER 3

TASK PERFORMED

All the tasks performed during the internship program were based on python, linux, networking essentials, databases and DevOps on cloud computing. The trainer had assigned some tasks which would prove to be quintessential for industry standards and easy understanding of the technology. These tasks were from IBM, Interleap((By Rooman Technologies) and Wadhwani Foundation

3.1 Python Programming-I

3.1.1 Introduction to Python and Basic Syntax

Python is a high-level, interpreted programming language widely used for its clean syntax and readability. It supports multiple programming paradigms including procedural, object-oriented, and functional programming. Python syntax emphasizes simplicity, using indentation to define blocks of code rather than braces or keywords. The basic syntax includes the use of variables, data types (such as strings, integers, floats, and booleans), operators, and comments. Python does not require explicit declaration of variable types, which makes it dynamically typed. Comments begin with a # symbol and are ignored by the interpreter.

3.1.2 Control Structures

Control structures manage the flow of execution within a Python program. The most common control structures include conditional statements and loops. Conditional statements (if, elif, else) execute certain blocks of code based on whether conditions are true or false.

Loops, such as for and while, are used to execute a block of code multiple times. A for loop iterates over a sequence (like a list or string), while a while loop runs as long as a specified condition remains true. Control statements like break, continue, and pass are also used within these structures to alter the flow.

3.1.3 Functions

Functions are reusable blocks of code that perform specific tasks. They help in modularizing code and making it more readable and maintainable.

Functions in Python are defined using the def keyword followed by the function name and parentheses, which may contain parameters. Functions can accept arguments and return values. They can also have default parameters and support keyword arguments. Python also supports anonymous functions using the lambda keyword, useful for small operations. Functions help avoid code duplication and allow logical segmentation of tasks.

3.1.4 Data Structures

Python includes several built-in data structures that are essential for organizing and storing data. These include:

- **Lists:** Ordered, mutable collections of items.
- **Tuples:** Ordered, immutable collections.
- **Dictionaries:** Key-value pairs used for fast lookups.
- **Sets:** Unordered collections of unique elements.

Each of these structures serves different purposes and is chosen based on the specific needs of the program. Python also provides functions and methods to manipulate these data structures effectively.

3.1.5 File Handling

File handling in Python allows programs to read from and write to files. This is crucial for tasks involving data storage, logging, configuration, or data exchange between applications.

Python uses the built-in open() function to interact with files. Files can be opened in different modes: read (r), write (w), append (a), and others. It is important to close files after operations to free system resources, though using a context manager (with statement) handles this automatically. File operations include reading data, writing data, checking for file existence, and handling exceptions during file access.

Based on above concepts, here is a simple python program example.

Example: Student Record Manager – A basic program that allows the user to add and display student records using functions, control structures, data structures, file handling, and follows proper Python syntax.

```python
# Student Record Manager

# Function to add a student record
def add_student(student_list):
    name = input("Enter student name: ")
    roll = input("Enter roll number: ")
    marks = float(input("Enter marks: "))

    student = {
        "name": name,
        "roll": roll,
        "marks": marks
    }
    student_list.append(student)
    print("Student added successfully!")

# Function to display all student records
def display_students(student_list):
    if not student_list:
        print("No records found.")
    else:
        print("\n--- Student Records ---")
        for student in student_list:
            print(f"Name: {student['name']}, Roll: {student['roll']},
                Marks: {student['marks']}")

# Function to save records to a file
def save_to_file(student_list):
    with open("students.txt", "w") as file:
        for student in student_list:
            line = f"{student['name']},{student['roll']},{student['marks
                ']}\n"
            file.write(line)
    print("Records saved to students.txt")

# Function to load records from file (if exists)
def load_from_file():
    student_list = []
    try:
        with open("students.txt", "r") as file:
            for line in file:
                name, roll, marks = line.strip().split(",")
                student = {
                    "name": name,
                    "roll": roll,
                    "marks": float(marks)
                }
                student_list.append(student)
    except FileNotFoundError:
        pass
    return student_list

# Main Program
students = load_from_file()

while True:
    print("\nMenu:")
    print("1. Add Student")
    print("2. Display Students")
    print("3. Save & Exit")

    choice = input("Enter your choice: ")

    if choice == '1':
        add_student(students)
    elif choice == '2':
        display_students(students)
    elif choice == '3':
        save_to_file(students)
        print("Exiting program. Goodbye!")
        break
    else:
        print("Invalid choice. Please try again.")
```

3.2 Python Programming-II

3.2.1 Modules and Libraries

Modules are individual Python files containing functions, classes, or variables that serve a specific purpose. They help in organizing code logically and improving reusability. Instead of writing large blocks of code in a single script, developers break it into multiple modules, each performing a particular task. Libraries are collections of modules packaged together to perform a broader range of functionalities. Python has a vast standard library (like math, datetime, etc.) as well as third-party libraries (such as NumPy, Pandas, Requests) which are used for tasks like scientific computing, data analysis, web development, etc. These libraries save time and effort by providing pre-written and optimized code for common tasks.

3.2.2 Exception Handling

In any software, errors during execution are inevitable — like dividing by zero, file not found, or invalid user input. Python uses exception handling to deal with such errors in a controlled way. Instead of crashing the program, Python allows developers to ”catch” these errors and define what should happen when an error occurs.

It enhances user experience and program reliability by allowing alternate actions, logging the error, or simply showing a user-friendly message. Exception handling is essential for writing secure and fault-tolerant applications, especially in areas like file operations, database access, and network communication.

3.2.3 Object-Oriented Programming (OOP)

OOP is a method of structuring a program using objects and classes. A class is a blueprint that defines the structure and behavior (attributes and methods), while an object is an instance of a class representing a specific entity.

This paradigm promotes key principles:

- **Encapsulation:** Keeping data and methods bundled together and protecting internal states.
- **Inheritance:** Allowing a class to inherit features from another class to promote code reuse.
- **Polymorphism:** Allowing methods to behave differently depending on the object using them.
- **Abstraction:** Hiding complex details and showing only the essential features.

OOP makes programs easier to manage, extend, and debug, especially as projects grow larger and more complex.

3.2.4 Python Programming with Databases

Python provides tools to connect and interact with various types of databases, such as SQLite, MySQL, and PostgreSQL. This integration allows developers to perform database operations like inserting, retrieving, updating, or deleting data directly from a Python program.

By using Database APIs (like sqlite3 for SQLite or external libraries like mysql-connector), developers can build powerful data-driven applications. This is particularly useful in areas like web development, data analysis, and enterprise software where persistent data storage and retrieval are crucial.

Proper use of Python with databases includes handling database connections, executing queries, managing transactions, and ensuring data security and integrity.

3.2.5 Flask Framework

Flask is a lightweight and beginner-friendly Python web framework used to develop web applications. It is classified as a micro-framework because it does not include default tools and libraries like database abstraction layers or form validation. This gives developers more control and flexibility.

Key Features:

- Minimal and Flexible: You can build applications from scratch with just the essentials, and extend them as needed.
- Built-in Development Server: Easy to test and run locally.
- Jinja2 Templating Engine: Allows creation of dynamic HTML content.
- Routing and Debugging Tools: Lets you define routes and debug during development.

3.2.6 Templates

Templates in Flask use HTML files with embedded dynamic content using Jinja2 syntax. This allows you to:

- Create a common structure for web pages (e.g., header, footer).
- Inject dynamic content like user data, lists, or conditions.
- Use loops (for) and conditionals (if) to control what is shown.
- Templates make your web application interactive and user-friendly by separating Python code from HTML design.

3.2.7 GitHub Codespace

GitHub Codespace is an online development environment integrated directly into GitHub. It provides:

- A cloud-hosted Visual Studio Code (VS Code) interface within your browser.
- Pre-configured tools and environments for development, such as Python or Flask.
- Real-time collaboration, code execution, and Git integration.

Steps for Setup:

1. Navigate to a provided Flask template repo.
2. Click “Code” > “Codespaces” > “Create Codespace on main”.
3. After 5–7 minutes, your development environment is ready.
4. You can run and modify Python files directly.

   Commit and push changes back to GitHub if needed.

3.2.8 Application Example: A Simple Inventory Management System

Imagine you’re building a desktop-based Inventory Management System for a small retail store. The system allows users (store employees) to add, update, search, and delete product information (like name, price, quantity) and store this data in a database.

1. **Modules and Libraries:** The application is divided into modules such as:

   - inventory_operations.py – handles all product-related logic
   - database_utils.py – handles database connection and query execution
   - main.py – runs the main application interface

   Libraries like sqlite3 (for database), os, and datetime are used to assist in operations and data handling.

2. **Exception Handling:** Handles cases such as:

   - Invalid user inputs (e.g., entering letters in a quantity field)
   - Database connection errors
   - Product not found errors during search

- Ensures that the program doesn't crash and gives helpful messages or logs errors for review.

3. **Object-Oriented Programming:** The system uses classes such as:

    - Product – represents a product with attributes like name, price, and quantity
    - InventoryManager – contains methods to add, update, delete, and search products.

    Using OOP helps organize the logic cleanly and makes it easy to extend later (like adding user authentication or reporting features).

4. **Python Programming with Databases:**

    - Product information is stored in an SQLite database.
    - SQL operations (INSERT, SELECT, UPDATE, DELETE) are used to manipulate data.
    - The database helps retain data even after the application is closed and reopened.

3.3 Prompt Engineering

3.3.1 Introduction to Prompt Engineering and AI Language Models

Prompt Engineering is the discipline of designing and refining textual inputs (prompts) to effectively interact with and control the outputs of AI language models such as OpenAI's GPT series, Google's PaLM, and Meta's LLaMA. These models are capable of generating human-like text, answering questions, writing code, creating summaries, and much more. Prompt engineering is essential to extract accurate, meaningful, and task-specific outputs from these models.

Key Concepts:

- AI Language Models are trained on extensive datasets and function by predicting the next word in a sequence. Over time, this ability allows them to perform tasks like text generation, translation, summarization, and even reasoning.

- Prompts are user-input queries or instructions given to these models. Their design strongly influences the quality, tone, and relevance of the AI's response.

- Goals of Prompt Engineering include maximizing output quality, maintaining ethical standards, and optimizing performance for specific tasks or domains.

Applications:

- Customer support bots
- Automated content generation
- Educational tools and tutoring
- Software development (code generation, debugging)
- Scientific research and analysis tools

3.3.2 Crafting Effective Prompts

Creating high-quality prompts is both an art and a science. The way a question or instruction is phrased can dramatically influence the response from an AI model. Crafting effective prompts ensures better alignment with desired outcomes.

Best Practices:

- **Clarity and Specificity:** Avoid vague wording. Define tasks explicitly (e.g., “Summarize this article in 3 bullet points”).
- **Contextual Framing:** Adding relevant background helps the AI understand the objective (e.g., “As a job recruiter, review this resume”).
- **Instructional Prompts:** Use direct instructions to guide behavior (e.g., “Write a professional email replying to a complaint”).
- **Few-shot Prompting:** Include 1–2 examples to show the model how to behave.
- **Chain of Thought Prompting:** Encourage step-by-step reasoning (e.g., “Let’s think through this step by step”).

Types of Prompts:

- **Zero-shot:** Ask a question without examples.
- **One-shot:** Give one example before the task.
- **Few-shot:** Provide multiple examples to guide the model.

3.3.3 Prompt Engineering for Specific Applications

Prompt engineering techniques vary depending on the domain and application. Tailoring prompts to suit the context of a specific task enhances model performance and reliability.

Domain-specific Examples:

- **Healthcare:** “Summarize this patient’s report in layman’s terms.”
- **Education:** “Generate quiz questions from this science article.”
- **Legal:** “Draft a rental agreement with standard legal clauses.”
- **Software Development:** “Generate Python code to sort a list using bubble sort.”

Techniques:

- **Role Assignment:** Direct the model’s tone and behavior by assigning roles (e.g., “Act as a data analyst…”).
- **Template-Based Prompting:** Use reusable prompt structures for repeated tasks.
- **Context Windows:** Manage the prompt size within model limits while maintaining essential information.

3.3.4 Ethical Considerations and Advanced Techniques

As language models grow in capability, ethical concerns become critical in prompt design. Engineers must ensure that outputs are non-biased, safe, and factually accurate.

Ethical Considerations:

- **Bias & Fairness:** Avoid prompts that reinforce stereotypes or generate biased content.
- **Safety:** Prevent misuse (e.g., generating harmful, illegal, or misleading outputs).
- **Privacy:** Avoid generating or requesting sensitive personal data.
- **Transparency:** Clearly communicate that content is AI-generated.

Advanced Prompting Techniques:

- **Self-refinement:** Ask the model to critique or improve its own response.
- **Memory Integration:** Use tools or APIs to preserve session context across interactions.

- **Multi-turn Prompts:** Design interactions that span multiple user inputs and responses.
- **Function Calling:** In advanced systems (e.g., OpenAI's API), use structured outputs like JSON for programmatic usage.

3.3.5 Hands-on Practice with AI Tools

Practical exposure to AI tools is crucial for mastering prompt engineering. It involves using real-world platforms to design, test, and iterate on prompts for various applications.

Popular AI Tools for Prompt Engineering:

- **OpenAI ChatGPT:** Create prompts, use APIs, explore GPT-3.5 and GPT-4 capabilities.
- **Google Gemini / Bard:** Experiment with Google's LLMs for different tasks.
- **Hugging Face Transformers:** Open-source models with customizable interfaces.
- **Anthropic Claude:** Known for its safety-first design and prompt understanding.
- **Prompt Engineering Platforms:** Tools like PromptLayer, FlowGPT, and LangChain offer testing, versioning, and sharing capabilities.

Practice Activities:

- **Prompt Tuning:** Experiment with modifying prompt wording to change output.
- **Multi-step Prompting:** Guide the AI through a sequence of tasks.
- **Debugging Outputs:** Analyze and refine prompts based on faulty or unexpected model responses.
- **Testing Across Models:** Compare prompt responses across different LLMs for consistency and quality.
- **Creating Prompt Templates:** Design reusable prompt formats for business or educational use.

3.4 AI Coding Assistants

3.4.1 AI-Powered Coding Assistants

AI-powered coding assistants are transforming the software development landscape by integrating artificial intelligence (AI), machine learning (ML), and natural language processing (NLP) to assist developers in writing, debugging, and optimizing code. These tools function as intelligent pair programmers, capable of understanding context, generating code snippets, and even completing entire functions based on natural language prompts. The rapid advancement of large language models (LLMs), such as OpenAI's GPT, Google's Gemini, and Meta's Code Llama, has significantly enhanced the capabilities of these assistants, making them indispensable in modern development workflows.

The primary advantage of AI coding assistants lies in their ability to boost developer productivity. By automating repetitive coding tasks, they allow programmers to focus on higher-level problem-solving and innovation. Additionally, these tools serve as valuable learning aids for beginners, offering real-time suggestions that align with best practices. However, their adoption also raises concerns, including potential over-reliance on AI-generated code and the need for careful validation to avoid errors or licensing issues.

3.4.2 AI-Powered Code Generation

One of the most groundbreaking features of AI coding assistants is their ability to generate functional code from natural language descriptions. Developers can simply describe what they need—for example, "a function to sort a list in descending order"—and the AI will produce the corresponding code in the desired programming language. This capability is powered by extensive training on vast repositories of open-source code, enabling the AI to recognize patterns and syntax across multiple languages, including Python, JavaScript, Java, and C++.

While AI-generated code accelerates development and reduces boilerplate, it is not without challenges. The quality of suggestions can vary, sometimes producing inefficient or insecure code. Developers must review and test AI-generated code thoroughly to ensure correctness and optimization. Furthermore, ethical considerations arise regarding code ownership and plagiarism, as AI models may inadvertently replicate licensed or proprietary code snippets. Despite these challenges, AI-powered code generation remains a game-changer, particularly in rapid prototyping and educational settings.

3.4.3 Debugging and Code Optimization with AI

AI coding assistants are increasingly being used to identify and fix bugs, optimize performance, and even predict potential vulnerabilities before they cause issues. Traditional debugging often requires manual inspection and extensive testing, but AI can analyze code statically and dynamically to detect anomalies, such as memory leaks, infinite loops, or incorrect logic. For instance, tools like Amazon Q and GitHub Copilot can suggest fixes for common errors or recommend more efficient algorithms.

Beyond debugging, AI assists in code optimization by analyzing execution paths and suggesting improvements. This includes refactoring redundant code, enhancing readability, and applying performance-enhancing techniques. Some advanced systems can even learn from a developer's coding style over time, offering personalized recommendations. However, while AI-driven debugging is powerful, human oversight remains crucial to ensure that automated suggestions align with the project's specific requirements and constraints.

3.4.4 Advanced Features of AI Coding Assistants

Modern AI coding assistants offer a range of advanced features that extend beyond basic code generation and debugging. These include:

- **Context-Aware Suggestions:** AI tools can analyze an entire codebase to provide relevant recommendations, ensuring consistency in naming conventions, design patterns, and architectural decisions.
- **Multi-Language Support:** Many assistants are trained on diverse programming languages, enabling seamless switching between tech stacks.
- **Integration with Development Environments:** Plugins for IDEs like VS Code, IntelliJ, and Eclipse allow real-time assistance without disrupting workflow.
- **Automated Documentation:** Some AI tools can generate inline comments or full documentation based on code structure, improving maintainability.
- **Security Scanning:** Advanced models can detect potential security flaws, such as SQL injection or cross-site scripting (XSS) vulnerabilities, and suggest mitigations.

These features collectively enhance the developer experience, reducing cognitive load and enabling faster, more reliable software development.

3.4.5 Hands-On Practice with GitHub Copilot, Amazon Q, and Google Gemini

To fully appreciate the capabilities of AI coding assistants, hands-on experience with leading tools like GitHub Copilot, Amazon Q, and Google Gemini is essential. GitHub Copilot, powered by OpenAI, excels in real-time code completion and is widely adopted due to its deep integration with GitHub repositories. Amazon Q, developed by AWS, focuses on enterprise-grade assistance, offering tailored support for cloud-based development and infrastructure management. Google Gemini, meanwhile, leverages Google's AI expertise to provide context-rich suggestions and multi-modal coding support.

Practical experimentation with these tools reveals their strengths and limitations. For example, GitHub Copilot is highly effective for rapid prototyping but may require fine-tuning for complex logic. Amazon Q shines in cloud-native development but has a steeper learning curve. Google Gemini's ability to process both code and natural language makes it versatile but sometimes less precise than specialized tools. By exploring these platforms, developers can determine which assistant best suits their workflow and project needs.

3.5 Linux Basics

3.5.1 Introduction And Basic Commands

At its core, Linux is an open-source operating system kernel, the fundamental layer that interacts directly with the computer's hardware. Unlike proprietary operating systems, Linux's source code is freely available, allowing for community-driven development and customization. This collaborative nature has led to the creation of numerous Linux distributions (often called ”distros”), each tailored for different purposes and user preferences. Popular distributions include Ubuntu, Fedora, Debian, CentOS, and Arch Linux, each with its own package management system and default configurations.

Interacting with the Linux system primarily occurs through the command-line interface (CLI), also known as the terminal or shell. This text-based interface allows users to execute commands to manage files, run programs, configure the system, and much more. Basic commands form the foundation of Linux interaction. For instance, **ls** lists the files and directories in the current location, while **cd (change directory)** navigates the file system. The **mkdir** command creates new directories, and **rm** is used to remove files and directories.

To view the contents of a file, commands like **cat, less**, or **more** are essential. Understanding these basic commands is crucial for navigating and manipulating the Linux environment effectively. Furthermore, concepts like absolute and relative paths are fundamental for specifying file and directory locations accurately within the hierarchical Linux file system.

3.5.2 User and File Permissions Management

In a multi-user environment, managing user accounts and controlling access to files and directories is paramount for system security and stability. Linux implements a robust permission system to achieve this. Each file and directory is associated with an owner (the user who created it), a group (a collection of users), and others (everyone else). For each of these categories, Linux defines three fundamental types of permissions: read (r), write (w), and execute (x). Read permission allows viewing the contents of a file or listing the contents of a directory. Write permission grants the ability to modify a file or create/delete files within a directory. Execute permission allows running a file as a program or entering a directory.

These permissions are typically represented in a symbolic format (e.g., **-rwxr-xr**–) or a numerical format (e.g., 754). The **chmod** command is used to modify these permissions, allowing administrators and users to control who can access and manipulate files and directories. Understanding user and group management is equally important. The **useradd** and **userdel** commands are used to create and delete user accounts, while **groupadd** and **groupdel** manage groups. The **chown** command changes the owner of a file or directory, and **chgrp** changes the associated group. Proper management of users, groups, and file permissions is essential for maintaining a secure and organized Linux system.

3.5.3 Process Management and Shell Scripting Basics

When you run a program on a Linux system, it becomes a process. Process management involves overseeing these running programs, including starting, stopping, monitoring, and prioritizing them. Each process is assigned a unique Process ID (PID). Commands like **ps** (process status) allow you to view the currently running processes, often with options to filter by user or display more detailed information. The **top** or **htop** commands provide a dynamic, real-time view of system processes, including CPU and memory usage.

To control processes, Linux provides commands like **kill**, which sends signals to processes. The most common signal is **SIGTERM** (usually the default), which politely asks a process to terminate.

For stubborn processes, **SIGKILL** (signal 9) can be used to forcefully terminate them. Backgrounding processes (running them without keeping the terminal occupied) is achieved using the **&** symbol.

Shell scripting involves writing a sequence of commands in a file, which the shell (like Bash, the default on many Linux systems) can then execute. This allows for automation of repetitive tasks, system administration, and the creation of custom tools. Basic shell scripting includes defining variables, using control flow structures (like **if** statements and **for** loops), and working with command-line arguments. Understanding how to pipe the output of one command to the input of another using **|** (pipes) and redirecting input and output using >, », and < are fundamental shell scripting concepts. Even basic shell scripts can significantly enhance efficiency and automate common Linux tasks.

3.5.4 Package Management and Disk Management

Managing software on a Linux system is handled through package management systems. These systems simplify the process of installing, upgrading, and removing software. Different Linux distributions use different package managers. For Debian-based systems like Ubuntu, apt (Advanced Package Tool) is commonly used with commands like **apt update** (to refresh the package lists), **apt install <package_name>** (to install a package), **apt upgrade** (to upgrade installed packages), and **apt remove <package_name>** (to uninstall a package). Red Hat-based systems like Fedora and CentOS use **dnf** (or the older **yum**), with similar commands like **dnf update**, **dnf install**, **dnf upgrade**, and **dnf remove**. Arch Linux uses **pacman**. Package managers also handle dependencies, ensuring that all necessary libraries and software components are installed for a program to function correctly.

Disk management involves organizing and utilizing the storage devices connected to the system. This includes partitioning disks (dividing them into logical sections), formatting partitions with file systems (like ext4, XFS, or Btrfs), and mounting file systems to make them accessible within the directory structure. Commands like f**disk**, **parted**, and **gparted** (a graphical tool) are used for partitioning. The **mkfs** command is used to create file systems on partitions. Mounting is done using the **mount** command, and the **/etc/fstab** file configures automatic mounting of file systems at boot time. Understanding disk space usage is also important, and commands like **df** (disk free) and **du** (disk usage) provide insights into how storage is being utilized. Logical Volume Management (LVM) is another advanced topic that provides more flexible disk management capabilities.

3.5.5 Networking Basics and System Monitoring

In today's interconnected world, understanding basic networking concepts in Linux is crucial. This includes configuring network interfaces (like Ethernet or Wi-Fi), understanding IP addresses, subnet masks, gateways, and DNS servers. Tools like **ip addr** (or the older **ifconfig**) are used to view and configure network interface settings. The **ping** command is used to test network connectivity to other hosts. Commands like **netstat** or **ss** can display network connections, routing tables, and interface statistics. Firewall management, often using **iptables** or **firewalld**, is essential for securing the system by controlling network traffic.

System monitoring involves keeping track of the system's performance and resource utilization. This includes monitoring CPU usage, memory usage (RAM and swap), disk I/O, and network traffic. Tools like **top**, **htop**, **vmstat** (virtual memory statistics), and **iostat** (I/O statistics) provide real-time and historical performance data. Log files, typically located in the **/var/log** directory, record system events, errors, and security information, making them invaluable for troubleshooting and monitoring system health. Understanding how to analyze these logs using tools like **grep** and **tail** is an important skill. System monitoring helps in identifying performance bottlenecks, detecting potential issues, and ensuring the overall stability and efficiency of the Linux system.

3.6 Networking Essentials

3.6.1 Introduction To Networking

At its core, networking is the practice of connecting two or more computing devices together so that they can communicate and share resources. These resources can include files, printers, internet access, and various applications. The scale of networks can vary dramatically, from a small home network connecting a few devices to the vast global network of the internet. The fundamental goal of networking is to enable efficient and reliable data exchange between connected entities.

The need for networking arose from the desire to share information and resources efficiently. In the early days of computing, data was often transferred physically using storage media. Networking provided a more convenient and faster way to access and share information. Different types of networks exist based on their geographical scope and purpose. A Personal Area Network (PAN) connects devices within a person's immediate vicinity, while a Local Area Network (LAN) connects devices within a limited geographical area like an office or home. A Metropolitan Area Network (MAN) covers a larger area, such as a city, and a Wide Area Network (WAN) spans large geographical distances, potentially connecting multiple LANs. The internet is the largest example of a WAN.

3.6.2 TCP/IP Model and IP Addressing

The Transmission Control Protocol/Internet Protocol (TCP/IP) model is a foundational suite of communication protocols used to interconnect network devices on the internet. It's a conceptual framework that organizes the complex process of network communication into four distinct layers:

- **Link Layer (or Network Interface Layer):** This is the lowest layer and deals with the physical transmission of data between two directly connected nodes. It encompasses technologies like Ethernet and Wi-Fi.
- **Internet Layer:** This layer is primarily responsible for addressing and routing packets across networks. The Internet Protocol (IP) operates at this layer, defining how data packets are addressed (using IP addresses) and routed from a source to a destination.
- **Transport Layer:** This layer provides reliable and ordered data delivery (TCP) or faster, less reliable delivery (UDP) between applications running on different hosts. TCP ensures that data arrives correctly and in the correct sequence, while UDP is often used for applications where speed is more critical than guaranteed delivery, such as streaming.
- **Application Layer:** This is the highest layer and interacts directly with end-user applications. It includes protocols like HTTP (for web browsing), SMTP (for email), and FTP (for file transfer).

IP addressing is a crucial aspect of the Internet Layer. An IP address is a logical numeric label assigned to each device participating in a computer network that uses the IP for communication between its nodes. There are two main versions of IP addresses: IPv4 and IPv6. IPv4 addresses are 32-bit numbers typically represented in dotted decimal notation (e.g., 192.168.1.100). Due to the explosive growth of the internet, the number of available IPv4 addresses has been exhausted, leading to the adoption of IPv6, which uses 128-bit addresses, offering a significantly larger address space. Understanding IP addressing, including the concepts of public and private IP addresses, is fundamental to grasping how devices are identified and communicate on a network.

3.6.3 Subnetting and DHCP

Subnetting is the practice of dividing a larger IP network into smaller, logical subnetworks or subnets. This is done to improve network organization, efficiency, and security.

By creating subnets, network administrators can segment network traffic, making it easier to manage and control. Subnetting involves manipulating the subnet mask, which is used to distinguish the network portion of an IP address from the host portion. The subnet mask is also a 32-bit number (for IPv4) and is often represented in dotted decimal notation (e.g., 255.255.255.0). The process of subnetting involves borrowing bits from the host portion of the IP address to create network bits, thus increasing the number of network segments but reducing the number of available host addresses within each subnet.

Dynamic Host Configuration Protocol (DHCP) is a network protocol that enables a server to automatically assign IP addresses and other network configuration parameters (such as the subnet mask, default gateway, and DNS server addresses) to devices on a network. This eliminates the need for manual configuration of IP addresses on each device, simplifying network administration and reducing the risk of IP address conflicts. When a DHCP-enabled device boots up, it sends a DHCP request to the DHCP server.

3.6.4 Application Layer Protocols

The Application Layer in the TCP/IP model is where end-user applications interact with the network. This layer utilizes various protocols to facilitate specific tasks and services. Some key application layer protocols include:

- **Hypertext Transfer Protocol (HTTP):** The foundation of data communication on the World Wide Web. It defines how web browsers and web servers communicate, enabling the transfer of web pages, images, and other content. HTTPS is a secure version of HTTP that uses encryption to protect the data being transmitted.
- **Domain Name System (DNS):** Translates human-readable domain names (like ”https: //www.google.com/search?q=google.com”) into IP addresses that computers use to identify each other on the network. DNS acts as a phonebook for the internet.
- **Simple Mail Transfer Protocol (SMTP):** The standard protocol for sending email messages between mail servers.
- **Post Office Protocol version 3 (POP3) and Internet Message Access Protocol (IMAP):** Protocols used by email clients to retrieve email from mail servers. IMAP allows users to access and manage their emails on the server from multiple devices,

while POP3 typically downloads emails to a single client and may delete them from the server.

- **File Transfer Protocol (FTP):** Used for transferring files between a client and a server. While still used, it's often being replaced by more secure protocols like SFTP (SSH File Transfer Protocol).
- **Secure Shell (SSH):** A cryptographic network protocol for securely operating network services over an unsecured network. It's commonly used for remote command-line access to servers.

Understanding these application layer protocols is crucial for comprehending how various internet services and applications function at a fundamental level. Each protocol has its specific purpose and defines the rules for communication between applications.

3.6.5 Network Troubleshooting and Basic Security

Network troubleshooting involves identifying, diagnosing, and resolving problems that occur within a network. This can range from simple connectivity issues to more complex performance problems. Common troubleshooting tools and techniques include:

- **Ping:** Used to test the reachability of a host on an IP network and measure the round-trip time for messages sent from the originating host to a destination computer.
- **Traceroute (or Tracert on Windows:)** Displays the path (routers) taken by packets across an IP network. It can help identify where network connectivity issues might be occurring.
- **IP Configuration Tools (e.g., ipconfig on Windows, ip addr on Linux):** Used to view the network configuration of a local machine, including IP address, subnet mask, and default gateway.
- **DNS Lookup Tools (e.g., nslookup, dig):** Used to query DNS servers to find the IP address associated with a domain name or vice versa.

- **Packet Sniffers (e.g., Wireshark):** Capture and analyze network traffic, allowing for detailed examination of communication between devices.

Basic network security involves implementing measures to protect a network and its data from unauthorized access, use, disclosure, disruption, modification, or destruction. Some fundamental security practices include:

- **Strong Passwords and Authentication:** Using complex and unique passwords for network devices and user accounts, and implementing secure authentication mechanisms.
- **Firewalls:** Act as a barrier between a trusted internal network and an untrusted external network (like the internet), controlling incoming and outgoing network traffic based on predefined rules.
- **Wireless Security (e.g., WPA3):** Securing wireless networks with strong encryption protocols to prevent unauthorized access.
- **Regular Software Updates and Patching:** Keeping network devices and software up to date with the latest security patches to address known vulnerabilities.
- **Awareness of Social Engineering and Phishing:** Educating users about common security threats and how to avoid them.

3.7 Databases

3.7.1 Introduction to SQL/NoSQL Databases

This topic provides a foundational understanding of database systems, distinguishing between SQL (Structured Query Language) and NoSQL (Not Only SQL) databases. SQL databases are relational, using structured schemas and tables to organize data. They are best suited for complex queries and transactional operations. In contrast, NoSQL databases are non-relational and designed for scalability and flexibility. They include document stores, key-value pairs, wide-column stores, and graph databases. The session typically introduces use cases for each and explains when to choose one over the other, depending on data structure, scalability needs, and consistency requirements.

3.7.2 SQL Querying

SQL querying focuses on interacting with relational databases using SQL. It includes the basics of SELECT, INSERT, UPDATE, and DELETE statements, as well as more advanced operations like JOIN, GROUP BY, ORDER BY, and nested queries. This section teaches how to retrieve and manipulate data effectively. Understanding SQL querying is crucial for tasks like generating reports, managing datasets, or feeding data into applications. Key skills also include filtering with WHERE, aggregating with functions like COUNT() or SUM(), and optimizing queries for performance.

3.7.3 Database Normalization & Indexing

Database Normalization is a systematic approach to organizing data in a relational database to reduce redundancy and improve data integrity. The process involves dividing large tables into smaller, related tables and defining relationships between them. It is typically carried out through a series of "normal forms" (1NF, 2NF, 3NF, etc.), each addressing specific types of anomalies:

- **NF (First Normal Form):** Ensures each column contains atomic (indivisible) values and each record is unique.
- **2NF (Second Normal Form):** Eliminates partial dependencies by ensuring that all non-key attributes are fully functionally dependent on the primary key.
- **3NF (Third Normal Form):** Removes transitive dependencies, ensuring that non-key attributes are not dependent on other non-key attributes.

Indexing, on the other hand, is a technique used to enhance database performance, particularly in speeding up data retrieval operations. An index is a data structure (typically a B-tree or hash table) that allows the database engine to find records more quickly than scanning the entire table.

Key points about indexing:

- Primary Indexes are created automatically on primary keys.
- Secondary (or Non-clustered) Indexes can be created on other columns to optimize query performance.
- Composite Indexes are created on multiple columns to handle queries involving multiple fields.

While indexes greatly speed up SELECT queries, they also come with trade-offs:

- Increased storage space.
- Slower insert/update/delete operations, as the index must also be updated.

3.7.4 NoSQL Databases

NoSQL databases are designed to handle unstructured, semi-structured, or rapidly changing data. Unlike relational databases, NoSQL systems are schema-less and can manage a variety of data types, making them ideal for big data and real-time applications. Common types include:

- Document databases (e.g., MongoDB) – store data in JSON-like documents.
- Key-value stores (e.g., Redis) – store data as a collection of key-value pairs.
- Column-family stores (e.g., Cassandra) – organize data by columns rather than rows.
- Graph databases (e.g., Neo4j) – focus on relationships between data points.

NoSQL databases are widely used in web development, social media, and IoT applications for their speed, scalability, and flexibility.

3.8 Web Development using Flask

3.8.1 Introduction to Flask and Application Setup

Flask is a micro web framework written in Python that provides the essential tools to build web applications quickly and with minimal overhead. It is called a ”micro” framework because it keeps the core simple and extensible while allowing developers to add plugins and extensions as needed. The setup process begins by installing Flask using a package manager like pip. Once installed, a basic Flask application typically starts with importing Flask, creating an app instance (app = Flask(__name__)), and defining routes using the @app.route() decorator. These routes map URLs to Python functions, also known as views. When the development server is run using flask run, the application becomes accessible via a web browser. This initial phase helps build an understanding of Flask’s architecture, request handling, and routing mechanism—key elements for building scalable web apps.

3.8.2 Working with Templates and Static Files

In Flask, the separation of logic and presentation is maintained using templates and static files. Templates are HTML files enriched with dynamic placeholders and control structures, rendered using Flask’s built-in Jinja2 templating engine. These templates allow the injection of Python variables (variable) and the use of logical constructs like loops and conditions (for item in list) directly within the HTML.

Flask also supports template inheritance, which promotes code reuse by enabling developers to define a base layout and extend it in child templates. Static files such as CSS, JavaScript, and images are stored in a designated /static folder and linked using the url_for('static', filename='...') function. Together, templates and static files enable developers to create visually appealing and interactive frontends while keeping the application logic clean and organized.

3.8.3 Handling Forms and User Input

Handling forms is a crucial aspect of web development as it enables user interaction with the application. In Flask, forms are typically created in HTML using standard elements like <input>, <textarea>, and <select>, and submitted via the POST method. On the backend, Flask uses the request object to capture user input (request.form['fieldname']). Validation is important at this stage to prevent incorrect or malicious data from being processed. For more complex forms and secure handling, Flask can be integrated with Flask-WTF, which provides CSRF protection and built-in validation features using Python classes.

3.8.4 Database Integration with Flask-SQLAlchemy

Flask-SQLAlchemy is an extension that simplifies working with databases in Flask applications by integrating SQLAlchemy ORM (Object Relational Mapper). It allows developers to define database schemas as Python classes (models), which are then translated into actual tables in a relational database. Each model represents a table, and each class attribute represents a column. For instance, a User class might have attributes like id, username, and email. The database connection is configured via a URI (usually in the Flask config), and CRUD (Create, Read, Update, Delete) operations can be easily performed using intuitive Python methods like db.session.add(), query.all(), or filter_by(). Flask-Migrate, another extension, is often used alongside Flask-SQLAlchemy to manage database migrations, ensuring that database schema changes are tracked and reproducible. This integration provides a structured and maintainable approach to data management in Flask apps.

3.8.5 Authentication and Deploying Flask Applications

Authentication is a fundamental requirement for most web applications, especially those that manage user data or provide personalized experiences. Flask provides several tools for implementing secure authentication. A common method involves using hashed passwords stored in a database and managing user sessions through Flask's session object. For more advanced functionality, the Flask-Login extension is used to handle user sessions, login/logout flow, and access control.

Once the application is complete, deploying it to a live server is the final step. Deployment involves preparing the application with production settings (disabling debug mode, setting environment variables), freezing dependencies in requirements.txt, and using platforms like Heroku, Render, or PythonAnywhere to host the app. Additionally, deployment might require a Procfile and WSGI configuration to serve the app efficiently. This stage ensures that the Flask application is accessible to end-users on the internet and is running in a secure, scalable, and reliable environment.

3.9 Introduction to Cloud Computing

3.9.1 Introduction to Cloud Computing,Core Cloud Services

Cloud computing delivers IT resources such as servers, storage, databases, networking, software, and analytics over the internet. It allows organizations to avoid the cost and complexity of owning and maintaining physical infrastructure. The three main service models are:

- **Infrastructure as a Service (IaaS):** Offers virtualized computing resources over the internet (e.g., AWS EC2, Azure Virtual Machines).
- **Platform as a Service (PaaS):** Provides a platform for developers to build and deploy applications without managing the underlying infrastructure (e.g., Google App Engine, Heroku).
- **Software as a Service (SaaS):** Delivers fully managed applications directly to users over the web (e.g., Google Workspace, Salesforce).

Core services in cloud computing include computing power, storage solutions (object and block storage), relational and NoSQL databases, content delivery networks (CDNs), and monitoring tools. These services are scalable, cost-effective, and designed for high availability.

3.9.2 Core Cloud Services

Compute services such as virtual machines and containerized environments (e.g., Docker, Kubernetes) offer flexible and scalable execution of applications. Serverless computing platforms like AWS Lambda and Azure Functions run code without provisioning servers.

Storage solutions include:

- **Object Storage:** Used for unstructured data (e.g., Amazon S3, Azure Blob Storage)
- **Block Storage:** Used for databases and file systems
- **File Storage:** Network-attached file shares for distributed environments

Database services cover both relational (e.g., MySQL, PostgreSQL) and NoSQL options (e.g., DynamoDB, Firestore). Networking tools manage virtual networks, IP addressing, load balancers, and secure connectivity via VPNs and firewalls. These services collectively enable the creation and management of reliable, secure, and scalable cloud-based applications.

3.9.3 Cloud DevOps

DevOps combines development and IT operations to automate and streamline the software delivery process. Cloud platforms support DevOps by offering tools for continuous integration and deployment (CI/CD), version control, infrastructure management, and monitoring.

Key tools and concepts:

- **CI/CD Pipelines:** Automate code building, testing, and deployment using services like GitHub Actions, Jenkins, and AWS CodePipeline.
- **Infrastructure as Code (IaC):** Manages cloud infrastructure through code using tools like Terraform and AWS CloudFormation.
- **Monitoring and Logging:** Tracks application performance and logs using tools such as Amazon CloudWatch and Azure Monitor.

DevOps in the cloud accelerates development cycles, ensures consistency in deployments, and improves operational efficiency.

3.9.4 Advanced Cloud Computing

Advanced cloud computing involves designing scalable, secure, and fault-tolerant systems. Key areas include:

- **Microservices Architecture:** Breaks applications into loosely coupled services managed and deployed independently.
- **Containerization and Orchestration:** Uses Docker for packaging applications and Kubernetes for orchestrating container clusters.
- **Auto-Scaling and Load Balancing:** Automatically adjusts resource capacity based on demand and distributes traffic across multiple servers.
- **Disaster Recovery and High Availability:** Ensures data backup, replication across regions, and quick recovery during failures.
- **Security and Identity Management:** Implements role-based access control, identity federation, and encryption standards using tools like IAM (Identity and Access Management).

3.10 Cloud Orchestration Platforms

3.10.1 Introduction to Cloud Orchestration - Kubernetes

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It organizes containers into pods, which are the smallest deployable units, and manages their lifecycle across a cluster of machines. Kubernetes provides powerful features like automatic scaling, self-healing, load balancing, and rolling updates, making it highly efficient for production-grade applications. It abstracts the underlying infrastructure, allowing developers to focus on application logic while Kubernetes handles networking, resource management, and fault tolerance. Kubernetes is platform-agnostic and can be deployed on public clouds, private data centers, or hybrid environments.

3.10.2 GKE (Google Kubernetes Engine)

GKE is Google Cloud's managed Kubernetes service, which simplifies the setup, management, and operation of Kubernetes clusters. It offers out-of-the-box scalability, security, and high availability. GKE automates tasks such as cluster provisioning, upgrades, monitoring, and node repair. It is tightly integrated with other Google Cloud services, enabling seamless networking, logging, and identity management. GKE supports auto-scaling, multi-zonal clusters, and node pools for managing different workloads efficiently. It is widely used for deploying containerized web applications, microservices, and ML workloads in a secure and scalable environment.

3.10.3 EKS (Elastic Kubernetes Service)

EKS is Amazon Web Services' managed Kubernetes platform, designed to run Kubernetes workloads in a scalable and secure manner. It offloads the operational overhead of managing the Kubernetes control plane to AWS, allowing developers to focus on building and running applications. EKS integrates with AWS IAM for fine-grained access control, VPC for networking, and services like CloudWatch for logging and monitoring. EKS supports both Fargate (serverless compute for containers) and EC2-based nodes, giving flexibility based on use case. It ensures high availability by distributing the control plane across multiple availability zones.

3.10.4 Azure Kubernetes Service (AKS)

AKS is Microsoft Azure's managed Kubernetes offering, designed to simplify Kubernetes cluster deployment and operations. It supports automatic upgrades, monitoring, and scaling, while maintaining security and compliance with Azure's native services. AKS integrates with Azure Active Directory for secure authentication and provides features like node auto-repair, private clusters, and virtual nodes for dynamic scaling. It is often used in enterprise environments for deploying scalable applications and microservices, with support for DevOps pipelines and CI/CD integrations using Azure DevOps or GitHub Actions.

3.10.5 Run Web Application on Each Cloud Platform

Deploying a web application across GKE, EKS, and AKS demonstrates the flexibility and portability of containerized workloads. The deployment typically involves containerizing the application using Docker, creating Kubernetes manifests (YAML files) for resources like pods, services, and deployments, and applying them to the respective clusters. Each platform provides CLI tools and dashboards for managing deployments:

- gcloud for GKE,
- aws eks and kubectl for EKS,
- az aks and Azure CLI for AKS.

Deployment also includes configuring ingress controllers or load balancers for external access and using secrets/config maps for managing environment-specific variables. This cross-platform experience highlights the uniformity Kubernetes provides while showcasing the unique features and integrations offered by each cloud provider.

3.11 Introduction to AWS, Amazon S3

3.11.1 AWS Account Creation and Introduction to IAM

Creating an AWS (Amazon Web Services) account involves registering with an email address, setting up billing information, and selecting a support plan. Once the account is created, AWS provides access to its web console and command-line interfaces. A critical part of account setup is the Identity and Access Management (IAM) service, which governs access control in AWS. IAM allows users to create and manage AWS users and groups, and use policies to define permissions for those entities. With IAM, organizations can enforce least privilege access, apply multi-factor authentication (MFA), and manage security credentials such as access keys. IAM roles can also be used to grant temporary access to services for users, applications, or services within AWS, promoting secure and scalable resource management.

3.11.2 Amazon S3 – Introduction

Amazon S3 (Simple Storage Service) is AWS's object storage service designed for storing and retrieving any amount of data from anywhere on the internet. S3 is highly scalable, durable (99.9% durability), and accessible via a web interface or APIs. Data is organized into buckets, and each object stored in S3 is assigned a unique key. Common use cases include backup and restore, static website hosting, data lake formation, and storing logs or analytics data. S3 supports multiple storage classes such as Standard, Intelligent-Tiering, Glacier, and Deep Archive, each optimized for specific access patterns and cost considerations. Objects can also have metadata and support versioning, which allows tracking of object modifications over time.

3.11.3 Amazon S3 – Advanced Features and Management

Beyond basic storage, Amazon S3 offers advanced features that support security, data management, and performance optimization. Key capabilities include:

- **Bucket Policies & Access Control Lists (ACLs):** Define granular access permissions.
- **Encryption:** Supports server-side encryption using S3-managed keys (SSE-S3), AWS KMS (SSE-KMS), or customer-provided keys (SSE-C).
- **Lifecycle Policies:** Automate data transition between storage classes or expiration of objects to optimize cost.
- **Cross-Region Replication (CRR):** Automatically replicates data to another AWS region for disaster recovery.
- **Event Notifications:** Trigger workflows (e.g., Lambda functions) upon events like object uploads or deletions.

These features enable organizations to build secure, automated, and scalable storage solutions tailored to business needs.

3.11.4 EC2 Fundamentals

Amazon EC2 (Elastic Compute Cloud) provides resizable compute capacity in the cloud. It allows users to launch virtual servers (known as instances) quickly and scale them according to application demand. Each EC2 instance can be configured with specific CPU, memory, storage, and network capabilities. Amazon provides various instance families (e.g., General Purpose, Compute Optimized, Memory Optimized) for different workloads.

Key EC2 fundamentals include:

- **AMI (Amazon Machine Image):** Pre-configured templates used to launch instances.
- **Security Groups:** Act as virtual firewalls for instances, controlling inbound and outbound traffic.
- **Elastic IPs:** Static IP addresses for dynamic cloud computing.
- **Key Pairs:** Used for secure SSH access to instances.

EC2 is foundational for hosting web servers, databases, enterprise applications, and performing batch processing.

3.11.5 EC2 Monitoring and Management

Efficient management and monitoring of EC2 instances are essential for performance optimization, cost control, and security. AWS provides multiple tools for these purposes:

- **Amazon CloudWatch:** Offers metrics, logs, and alarms to monitor EC2 instances in real time, such as CPU utilization, disk I/O, and network traffic.
- **AWS Systems Manager:** Provides visibility and control of the infrastructure, enabling automation of operational tasks like patch management, compliance scanning, and remote command execution.
- **Auto Scaling Groups (ASGs):** Automatically adjust the number of EC2 instances based on demand, ensuring high availability and efficient cost usage.
- **Elastic Load Balancing (ELB):** Distributes incoming application traffic across multiple instances to enhance fault tolerance and availability.

Through these tools, administrators can ensure applications are running efficiently while minimizing manual intervention.

3.12 AWS Command Line Interface (CLI) Basics

3.12.1 Install and Configure the AWS CLI on Windows and Apple/Linux

The AWS CLI (Command Line Interface) is a powerful tool that enables users to interact with AWS services through terminal commands, providing automation and scripting capabilities. Installation is platform-specific: for Windows, users download and run the MSI installer; on macOS and Linux, it can be installed using package managers like brew or apt. After installation, configuration is done using the aws configure command, where the user inputs their Access Key ID, Secret Access Key, default region, and output format. These credentials are stored in a hidden folder (/.aws/credentials), and they allow secure communication with AWS APIs. This setup enables access to AWS services directly from the terminal, streamlining deployment and management tasks.

3.12.2 AWS CLI to Build and Manage Simple AWS Systems

The AWS CLI can be used to create and manage core AWS services such as EC2, S3, and IAM using simple commands. For example:

- Launching an EC2 instance using aws ec2 run-instances
- Uploading files to S3 using aws s3 cp
- Creating IAM users or policies with aws iam create-user and aws iam put-user-policy

This approach is ideal for scripting repetitive tasks, deploying infrastructure without a graphical interface, and integrating into automated DevOps pipelines. It is also helpful for developers who need to test services in a lightweight, fast manner. Additionally, CLI commands support output formatting in JSON, text, or table views, making it suitable for both human readability and automation scripts.

3.12.3 AWS CLI for EB Tools (Elastic Beanstalk Tools)

Elastic Beanstalk CLI (EB CLI) is a component built on top of AWS CLI to manage applications deployed via AWS Elastic Beanstalk, a Platform-as-a-Service (PaaS). The EB CLI simplifies deployment and management of web applications. Developers can:

- Initialize projects using eb init.
- Create environments with eb create.
- Deploy updates using eb deploy.
- Monitor application status with eb status and eb health.

The EB CLI automates many manual AWS console actions and integrates well with CI/CD pipelines, allowing teams to continuously deploy updated application code with minimal friction. It abstracts away infrastructure-level details, enabling developers to focus on application logic rather than server management.

3.12.4 AWS CLI for SAM Tools (Serverless Application Model)

AWS SAM (Serverless Application Model) CLI extends AWS CLI capabilities to manage serverless applications. It is tailored to work with Lambda functions, API Gateway, DynamoDB, and other services used in serverless architectures. Key capabilities include:

- Building and packaging applications with sam build and sam package.
- Local testing with sam local invoke and sam local start-api.
- Deployment using sam deploy with support for change sets.

SAM CLI provides a development workflow similar to traditional applications, allowing local testing and debugging before deploying to the cloud. It bridges the gap between local development and cloud-based execution of serverless applications.

3.12.5 AWS CLI for Templated Files

The AWS CLI supports usage of templated files such as JSON or YAML configuration files for managing complex resource definitions and actions. These templates are often used in combination with services like CloudFormation or SAM, allowing users to define infrastructure as code. Users can run commands like aws cloudformation deploy or aws s3api put-bucket-policy –policy file://policy.json to apply configurations.

Templating enables repeatability and version control for infrastructure, making environments easy to replicate and audit. It also supports parameterization, allowing a single template to be reused with different input values, ideal for staging, development, and production setups.

3.13 DevOps on AWS

3.13.1 DevOps Methodologies of Culture, Practices, and Tools

DevOps is a combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity. It bridges the gap between development and operations teams, aiming for faster software releases, better collaboration, and improved deployment reliability.

- **Culture:** Emphasizes shared responsibility, collaboration, ownership, continuous learning, and feedback.
- **Practices:** Continuous Integration (CI), Continuous Delivery (CD), Infrastructure as Code (IaC), monitoring and logging, automated testing, and agile project management.
- **Tools:** Tools support automation across the software delivery lifecycle. Examples include Git for source control, Jenkins or AWS CodeBuild for CI, AWS CodePipeline for orchestration, and Terraform or AWS CloudFormation for IaC.

The core goal is to create a streamlined, automated, and responsive software delivery pipeline that enhances innovation without compromising stability or security.

3.13.2 Amazon’s Transformation to DevOps

Amazon’s own shift to DevOps serves as a model of scalability and operational excellence. Initially operating on a monolithic architecture, Amazon migrated to a service-oriented architecture (SOA), eventually evolving into microservices hosted on cloud infrastructure.

Key aspects of Amazon’s transformation:

- **Decentralized ownership:** Teams are organized as ”two-pizza teams” (small, autonomous groups), each owning a specific service end-to-end.
- **Infrastructure Automation:** Heavy adoption of infrastructure automation tools and auto-scaling enabled rapid deployments and high availability.
- **Cultural Shift:** Engineering teams embraced “You build it, you run it” mindset, leading to faster issue resolution and more resilient services.
- **Tooling & Monitoring:** Amazon built robust internal tooling for CI/CD, deployment, logging, and performance monitoring, many of which evolved into AWS services.

This transformation drastically reduced deployment times, enhanced customer experience, and formed the foundation of AWS’s modern DevOps ecosystem.

3.13.3 Categorize and Describe Key AWS DevOps Services That Support the Application Lifecycle

AWS offers a comprehensive suite of services that span the entire DevOps application lifecycle. In the planning and tracking phase, AWS CodeStar plays a significant role by providing a unified interface to manage software development activities, integrate with AWS services, and enable collaboration among teams. For the development and build phases, AWS CodeCommit serves as a secure, scalable Git-based repository that allows teams to manage source code efficiently, while AWS CodeBuild handles the automated compiling of source code, running of tests, and creation of deployment-ready artifacts in a fully managed, continuous scaling environment.

Testing can also be integrated directly into the pipeline, with CodeBuild executing unit tests and third-party tools providing options for integration, security, or performance testing. For release and deployment, AWS CodePipeline automates the entire software release process through a series of stages like build, test, and deploy, while AWS CodeDeploy ensures smooth delivery of applications to compute services such as EC2, AWS Lambda, or on-premises infrastructure. During the operation and monitoring stages, Amazon CloudWatch offers deep visibility into application logs, metrics, and performance dashboards, helping detect and respond to operational issues. For distributed applications, AWS X-Ray provides detailed insights into service requests, aiding in debugging and performance optimization. These services are designed for seamless integration, enabling rapid iteration, continuous improvement, and robust automation across development workflows.

3.13.4 Identify the AWS Services Used to Automate the Continuous Integration and Continuous Delivery (CI/CD) Process

AWS provides fully managed services to automate each stage of the Continuous Integration and Continuous Delivery (CI/CD) process, enhancing agility, reliability, and control over software delivery. The CI/CD process begins with source control, handled effectively by AWS CodeCommit, which offers secure, private Git repositories to manage code versioning and collaboration. For continuous integration, AWS CodeBuild compiles code, executes automated tests, and generates output artifacts. This ensures that every code change is verified in isolation, reducing integration issues and supporting fast feedback loops.

The continuous delivery aspect is streamlined through AWS CodePipeline, which automates the orchestration of the pipeline from source through build, test, and deployment stages. When it comes to deployment, AWS CodeDeploy manages the delivery of application updates to multiple target environments including EC2, AWS Lambda, and ECS, with strategies to minimize downtime and support rollbacks. For testing, developers can incorporate custom test suites or third-party testing tools directly into the pipeline to maintain quality and security standards. Monitoring is achieved through Amazon CloudWatch, which collects logs, metrics, and alarms, and AWS Config, which assesses compliance and tracks configuration changes. These services collectively enable fully automated CI/CD pipelines that are code-driven, scalable, and easily managed through infrastructure-as-code practices.

3.13.5 Create and Control a CI/CD Pipeline

Creating and managing a CI/CD pipeline in AWS involves designing a series of automated stages that move code seamlessly from development to production. The pipeline begins with the Source stage, where code changes in repositories such as AWS CodeCommit or GitHub trigger the pipeline automatically. These triggers initiate the Build stage, where AWS CodeBuild compiles the codebase, runs unit tests, and produces build artifacts. Following this, the Test stage executes additional automated tests such as integration or security testing to validate the integrity and functionality of the application.

An optional Approval stage can be included before deployment, where a manual review is required to approve production releases, adding an extra layer of control. Finally, in the Deploy stage, AWS CodeDeploy manages the deployment of the application to target environments like EC2, ECS, or Lambda. The entire pipeline can be monitored and controlled using triggers and webhooks that automatically respond to code commits. Environment variables and configuration parameters are used to differentiate between development, staging, and production environments. Rollback strategies such as blue/green deployments or canary releases ensure reliability and minimize risks during deployments. These features make AWS pipelines robust, scalable, and suitable for modern DevOps practices in any organization.

3.14 Pre-Project Development

3.14.1 Design Thinking and MURAL

Design Thinking is a human-centered, iterative approach used for creative problem-solving. It encourages innovation by deeply understanding the user's needs, challenging assumptions, and redefining problems. The process emphasizes empathy and experimentation and is widely used across disciplines like engineering, business, and healthcare.

Phases of Design Thinking:

1. **Empathize:** Understand the user's needs through observation and engagement. Tools include interviews, personas, and empathy maps.
2. **Define:** Clearly articulate the problem by analyzing the gathered insights.
3. **Ideate:** Generate a wide range of possible solutions using brainstorming techniques, ”how might we” questions, and SCAMPER.
4. **Prototype:** Develop scaled-down versions or mockups of the product or idea.
5. **Test:** Trying out prototypes with users, gather feedback, and refine solutions.

MURAL Tool:
MURAL is a collaborative online whiteboard platform used for visual thinking and brainstorming. It enables remote teams to work together on problem-solving, planning, and creative design. You used MURAL for brainstorming, defining ideas, and aligning team goals visually during the internship .

3.14.2 Agile Methodology (IBM Template)

Agile Fundamentals:

Agile is a software development methodology that promotes continuous iteration of development and testing throughout the software development lifecycle. Unlike traditional methods, Agile emphasizes flexibility, collaboration, and customer feedback.

Key Features of Agile:

- **Iterative Development:** Work is divided into small cycles called sprints.
- **Daily Stand-Ups:** Short meetings to track progress and blockers.
- **Roles:** Includes Product Owner, Scrum Master, and Development Team.
- **Artifacts:** Product backlog, sprint backlog, and increment.
- **Tools:** Agile tools like Kanban boards and Jira (if applicable) to manage tasks, user stories, and deadlines.

3.14.3 GitHub and Version Control

GitHub is a cloud-based platform that uses Git for version control, enabling multiple developers to collaborate on projects efficiently. You used GitHub for storing, managing, and tracking code and documents during your internship.

Key Concepts Applied:

- **Repositories:** Centralized folders containing all project files.
- **Commits & Branching:** Creating branches for individual features and merged them into the main branch after testing.
- **Pull Requests:** Used to propose and review code changes.
- **Collaboration:** Using GitHub Issues and Projects to manage and assign tasks across the team.

Impact: GitHub streamlined collaboration, reduced code conflicts, and enabled smooth version tracking.

3.15 Prerequisites of DevOps

3.15.1 SDLC Concepts

DevOps is a modern software engineering culture and practice that integrates software development and IT operations into a unified process. It emphasizes collaboration between traditionally siloed roles, such as developers and system administrators, to improve productivity and software quality. The approach encourages automation across all stages of the software development lifecycle—from planning and development to deployment and monitoring. By aligning teams and streamlining processes, DevOps aims to deliver software faster and more reliably while also ensuring stability and scalability in production environments. A typical DevOps workflow incorporates planning tools like Jira, source code repositories such as GitHub, continuous integration servers like Jenkins, configuration management with Ansible, and deployment tools including Docker and Kubernetes.

3.15.2 CI/CD Operations

Continuous Integration (CI) and Continuous Delivery (CD) form the backbone of DevOps practices. CI involves regularly merging code changes from multiple developers into a central repository, followed by automated builds and tests. This helps in identifying errors early and ensures that the codebase remains in a deployable state. Continuous Delivery builds upon CI by automating the release process up to the production environment, though it may still include manual approval gates. On the other hand, Continuous Deployment takes it a step further by pushing every successful code change directly into production, eliminating the need for manual intervention. These practices reduce integration issues, accelerate release cycles, and maintain software quality with consistency.

3.15.3 CI/CD Testing and Deployment

In the CI/CD process, testing plays a critical role. Automated testing methods such as unit tests, integration tests, and system tests ensure that the application functions as intended at every stage of development. Automation tools like Selenium and JUnit are widely used to implement these tests within the pipeline. The deployment process typically includes creating build artifacts, preparing the target environment, containerizing the application using Docker, and orchestrating containers with platforms like Kubernetes. Two common deployment strategies include Blue-Green Deployment, where traffic is switched between two identical environments to minimize downtime, and Canary Deployment, where updates are gradually rolled out to a small subset of users before full deployment.

3.16 Virtualization and Containerization

3.16.1 Virtualization

Virtualization and containerization are essential technologies in modern software development, offering improved resource efficiency, scalability, and deployment flexibility. Virtualization allows multiple operating systems to run on a single physical machine using virtual machines (VMs), each with its own OS and allocated resources. Containerization, on the other hand, packages applications and their dependencies into lightweight, isolated containers that share the host OS kernel, ensuring consistency across different environments.

Virtualization can be categorized into several types, including server virtualization, desktop virtualization, storage virtualization, and network virtualization. Server virtualization divides a physical server into multiple VMs, while desktop virtualization enables remote access to centralized desktop environments. Storage virtualization combines storage devices into a single manageable unit, and network virtualization merges hardware and software resources into a software-defined network.

The core components of virtualization include the hypervisor, which manages VMs, and the virtual machines themselves. Hypervisors come in two types: Type 1 (bare-metal) runs directly on hardware, and Type 2 (hosted) operates on an existing OS. Virtual machines mimic physical computers, providing isolated environments for applications.

Advantages of Virtualization:

- **Resource Efficiency:** Maximizes hardware utilization by running multiple VMs on a single server.
- **Scalability:** Allows dynamic allocation of resources based on demand.
- **Isolation:** Ensures security and stability by separating VMs.
- **Cost Savings:** Reduces hardware, energy, and space requirements.

3.16.2 Containerization and Docker:

Containerization is a lightweight alternative to virtualization, where applications run in isolated environments called containers. Unlike VMs, containers share the host OS kernel, making them faster and more resource-efficient. Docker is a leading containerization platform that simplifies the creation, deployment, and management of containers.

Docker Architecture: Docker consists of several key components:

- **Docker Engine:** The core runtime that manages containers, images, and networks.
- **Images:** Immutable templates used to create containers.
- **Containers:** Runtime instances of images.
- **Docker Hub:** A registry for sharing and storing Docker images.

3.16.3 Deploying Applications with Docker

Steps to Deploy an HTML File Using Docker:

1. **Create a Project Directory:** Organize your HTML file and related assets in a dedicated folder.
2. **Write a Dockerfile:**

```
            FROM nginx:alpine\\
            COPY index.html /usr/share/nginx/html\\
            EXPOSE 80
```

3. **Build the Docker Image:** Run the following command in the project directory:

```
         docker build -t html-server
```

4. **Run the Container:** Start the container with port mapping:

```
        docker run -d -p 8080:80 --name my-html-container html-
           server
```

5. **Access the Application:**
   Open a web browser and navigate to http://localhost:8080 to view the HTML file

3.16.4 Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is the practice of managing and provisioning infrastructure through machine-readable configuration files, enabling automation, consistency, and version control. IaC eliminates manual processes, reducing errors and improving efficiency.

Popular IaC Tools:

- **Terraform:** A declarative tool for multi-cloud infrastructure provisioning.
- **AWS CloudFormation:** AWS-native service for managing cloud resources.
- **Ansible:** A configuration management tool for automating server setups.

3.17 DevOps Automation Tools

3.17.1 Jenkins Sequential Pipelines

Creating a sequential pipeline in Jenkins allows for controlled execution of jobs with user prompts between stages. This is particularly useful for CI/CD workflows where manual approval is required before proceeding to the next step.

Below is a structured guide to setting up such a pipeline:

Steps to Create a Sequential Pipeline

1. **Install Jenkins:**
   - Download the Jenkins installer for Windows and complete the setup by unlocking Jenkins using the initial admin password.
   - Install suggested plugins and create an admin user.
2. **Create Freestyle Jobs:**
   - Define individual jobs (e.g., Job1, Job2, Job3) with simple commands like echo ”Job completed”.
3. **Pipeline Script:** Use a declarative pipeline script to sequence jobs with input prompts:

```
pipeline {
    agent any
    stages {
        stage('Job 1') {
            steps {
                build job: 'Job1', wait: true
                input message: 'Proceed to Job 2?'
            }
        }
        stage('Job 2') {
            steps {
                build job: 'Job2', wait: true
                input message: 'Proceed to Job 3?'
            }
        }
    }
}
```

4. **Execution:** Run the pipeline manually and approve each stage via the Jenkins interface.

3.17.2 Setting Up Jenkins Freestyle Projects

Steps to setup Jenkins Freestyle Project

1. **Create a Project:** Navigate to New Item > Freestyle Project.
2. **Configure Build Steps:** Add commands (e.g., java -version) under Execute Windows batch command.
3. **Run and Monitor:** Trigger builds manually and review console output for errors.

Example:

```

    java -version
    dir
```

3.17.3 Git and Git Bash Installation

Git is a widely used version control system that allows developers to track changes in their codebase, collaborate with team members, and manage different versions of a project efficiently. Git Bash, a command-line interface for Git on Windows, provides a Unix-like terminal experience, enabling users to run Git commands seamlessly.

Installation Process(For Windows OS)

1. **Download the Installer:** Visit the official Git website (https://git-scm.com/downloads) and download the Windows installer. Ensure you select the appropriate version (64-bit or 32-bit) based on your system requirements.
2. **Run the Installer:** Double-click the downloaded .exe file to launch the setup wizard. Accept the GNU General Public License agreement to proceed.

3. **Select Installation Components:** During installation, choose the components you wish to include. At a minimum, ensure Git Bash and Git GUI are selected. Additional options like Git LFS (Large File Support) and Associate .sh files with Bash can also be enabled for extended functionality.

4. **Configure the PATH Environment:** When prompted, select the recommended option: ”Git from the command line and also from 3rd-party software.” This ensures Git commands are accessible from both Git Bash and the Windows Command Prompt.

5. **Set Up SSH and Line Endings:** For secure connections, choose ”Use bundled OpenSSH” as the default SSH client. Configure line endings by selecting ”Checkout Windows-style, commit Unix-style line endings” to maintain compatibility across different operating systems.

6. **Choose the Terminal Emulator:** Opt for MinTTY as the default terminal for Git Bash, as it offers better features like resizable windows and Unicode support compared to the standard Windows console.

7. **Complete the Installation:** Once all configurations are set, proceed with the installation. After completion, check the box to ”Launch Git Bash” to open the terminal and verify the installation by running git –version.

Post-Installation Setup

After installing Git, configure your username and email to associate your commits with your identity:

```
    git config --global user.name "Your Name"
    git config --global user.email "your@email.com"
```

3.17.4 Minikube: Local Kubernetes Cluster for Development and Testing

Minikube is a lightweight, open-source tool designed to run a single-node Kubernetes cluster locally, making it ideal for development, testing, and learning Kubernetes without needing a full-scale cloud-based cluster. It creates a virtual machine (VM) on your local machine, deploying a minimal Kubernetes environment that includes core components like the API server, scheduler, controller manager, and kubelet. Minikube supports multiple container runtimes, including Docker, containerd, and CRI-O, allowing developers to test containerized applications in a Kubernetes-like environment before deploying to production.

Minikube simplifies Kubernetes adoption by providing a sandbox where users can experiment with deployments, services, and Helm charts. It integrates seamlessly with kubectl, the Kubernetes command-line tool, enabling users to manage pods, services, and deployments just as they would in a production cluster. Additionally, Minikube supports add-ons like the Kubernetes dashboard, ingress controllers, and metrics servers, enhancing its functionality. By simulating a real Kubernetes cluster locally, Minikube accelerates the development lifecycle, reduces infrastructure costs, and helps teams debug and optimize applications before moving to cloud-based Kubernetes services like EKS, GKE, or AKS.

3.17.5 Building a CI/CD Pipeline in GitLab: Automation from Code to Deployment

GitLab provides a robust, integrated CI/CD platform that automates the software development lifecycle—from code commits to production deployment. A GitLab pipeline is defined in a .gitlab-ci.yml file, where developers specify stages (e.g., build, test, deploy) and jobs (e.g., compile code, run unit tests, deploy to staging). GitLab Runners execute these jobs in isolated Docker containers or virtual machines, ensuring reproducibility and scalability.

1. **Build:** Compiles source code into executable artifacts (e.g., Docker images, binaries).
2. **Test:** Runs automated unit, integration, and security tests (e.g., using JUnit, Selenium, or OWASP ZAP).
3. **Deploy:** Pushes validated artifacts to environments (e.g., staging, production) using Kubernetes, AWS, or Azure.
4. **Monitor:** Tracks application performance with built-in observability tools or integrations like Prometheus.

GitLab's Auto DevOps feature further simplifies pipeline creation by automatically detecting languages and applying best practices. Advanced features include merge request pipelines, environment-specific variables, and manual approval gates for production deployments. By unifying version control, CI/CD, and monitoring, GitLab streamlines collaboration, reduces manual errors, and accelerates delivery.

3.17.6 ML Model Deployment: From Training to Production

Machine Learning (ML) model deployment involves packaging a trained model into a scalable, production-ready service that can serve predictions via APIs or batch processing. The process includes:

1. **Model Training:** Using frameworks like TensorFlow, PyTorch, or scikit-learn, models are trained on historical data and validated for accuracy.
2. **Packaging:** Models are serialized (e.g., as .pkl or .onnx files) and containerized using Docker for consistency across environments.
3. **Serving:** Deployed via:
   - **REST APIs:** Using Flask, FastAPI, or specialized tools like TensorFlow Serving.
   - **Serverless Platforms:** AWS Lambda or Google Cloud Functions for event-driven predictions.
   - **Kubernetes:** For scalable, resilient deployments with autoscaling (e.g., using KFServing or Seldon Core).
4. **Monitoring:** Tools like Prometheus or MLflow track model drift, latency, and accuracy, triggering retraining if performance degrades.

MLOps practices, such as versioning models with DVC and automating retraining pipelines, ensure continuous improvement. Challenges include managing GPU resources, A/B testing models, and ensuring low-latency inference.

3.17.7 IBM Cloud DevOps Services: Enterprise-Grade CI/CD and Automation

IBM Cloud DevOps Services offers a suite of tools for automating software delivery, leveraging IBM's cloud infrastructure and open-source technologies. Key components include:

1. Toolchain: Integrates Git repositories (GitHub, GitLab), CI/CD pipelines, and monitoring tools into a unified workflow.
2. Continuous Delivery: Uses Tekton-based pipelines to build, test, and deploy applications across IBM Cloud Kubernetes Service (IKS), OpenShift, or VMs.
3. DevOps Insights: Provides analytics for pipeline performance, security vulnerabilities (via IBM Cloud Security Advisor), and compliance checks.
4. Infrastructure as Code (IaC): Supports Terraform and Ansible for provisioning cloud resources programmatically.

CHAPTER 4

ASSIGNMENTS

4.1 Assignment 1: Setting Up AWS Services

4.1.1 Task 1: Create an AWS Free Tier Account

Step-by-Step Instructions:

1. Visit the AWS Website
    - Go to https://aws.amazon.com/ and click ”Create an AWS Account”.
2. Enter Account Details
    - Provide your email address, choose a unique AWS account name, and click ”Verify email address”.
    - Check your email for a verification code and enter it when prompted.
3. Set Password & Account Type
    - Create a strong password for your AWS root account.
    - Select ”Personal” or ”Business” account type.
4. Add Billing Information
    - Enter credit/debit card details (AWS may charge a small verification amount, which is refunded).
    - Fill in the required personal/business details.
5. Identity Verification
    - AWS may require a phone number for verification via SMS or call.
6. Select a Support Plan
    - Choose the ”Basic (Free)” plan unless you need paid support.
7. Sign In to AWS Console
    - Once verified, log in at https://console.aws.amazon.com/.

4.1.2 Task 2: Create Your First EC2 Instance

Step-by-Step Instructions

1. Log in to AWS Console
   - Navigate to EC2 Dashboard under ”Services” > ”Compute” > ”EC2”.
2. Launch an Instance
   - Click ”Launch Instance” and provide a name (e.g., MyFirstEC2).
3. Choose an Amazon Machine Image (AMI)
   - Select a Free Tier eligible AMI (e.g., Amazon Linux 2023 AMI or Ubuntu Server).
4. Select Instance Type
   - Choose t2.micro (Free Tier eligible).
5. Create a Key Pair
   - Generate a new key pair (e.g., my-key-pair.pem) and download it securely.
6. Configure Network Settings
   - Allow SSH (port 22) for initial access.
   - (Optional) Enable HTTP/HTTPS if hosting a website.
7. Launch the Instance
   - Click ”Launch Instance” and wait for it to initialize.
8. Connect to the Instance
   - Use SSH (for Linux/macOS) or PuTTY (Windows) with the downloaded key:

```
    ssh -i "my-key-pair.pem" ec2-user@<Public-IP>
```

Verification

- Check instance status in the EC2 dashboard.
- Run basic commands (e.g., ls, pwd) after connecting via SSH.

4.1.3 Task 3: Create an S3 Bucket and Upload Files

Step-by-Step Instructions

1. Open S3 Dashboard
   - Go to ”Services” > ”Storage” > ”S3”.
2. Create a Bucket
   - Click ”Create bucket”, enter a globally unique name (e.g., *my-bucket-123*).
   - Select a Region (e.g., *us-east-1*).
3. Configure Permissions
   - Uncheck ”Block all public access” if hosting public files (else keep it restricted). Click ”Create bucket”.
4. Upload Files
   - Select the bucket, click ”Upload”, and drag/drop files (e.g., images, documents).
   - Click ”Upload” to complete the process.
5. Access Files: To share a file publicly:
   - Select the file > ”Actions” > ”Make public”.
     Copy the Object URL (e.g., https://my-bucket-123.s3.amazonaws.com/file.txt).

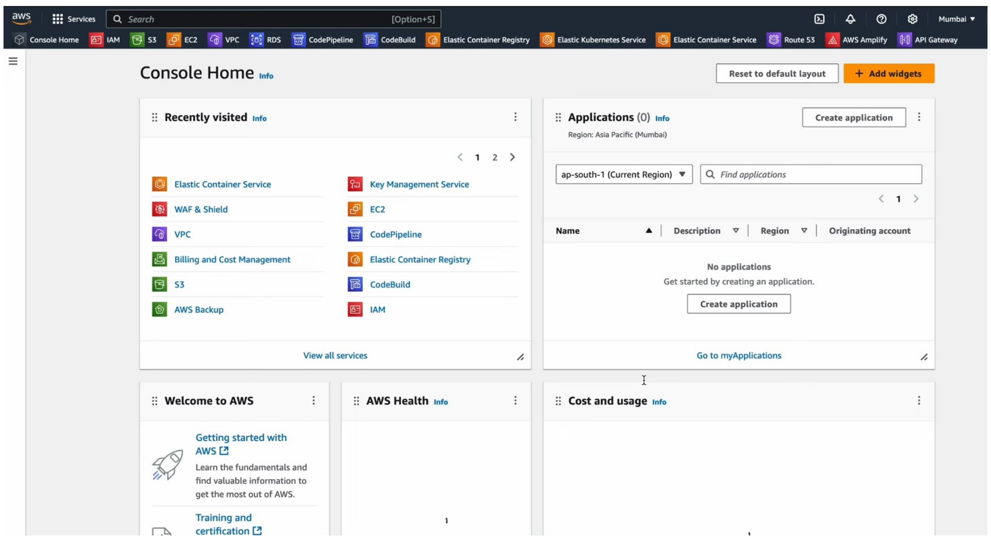
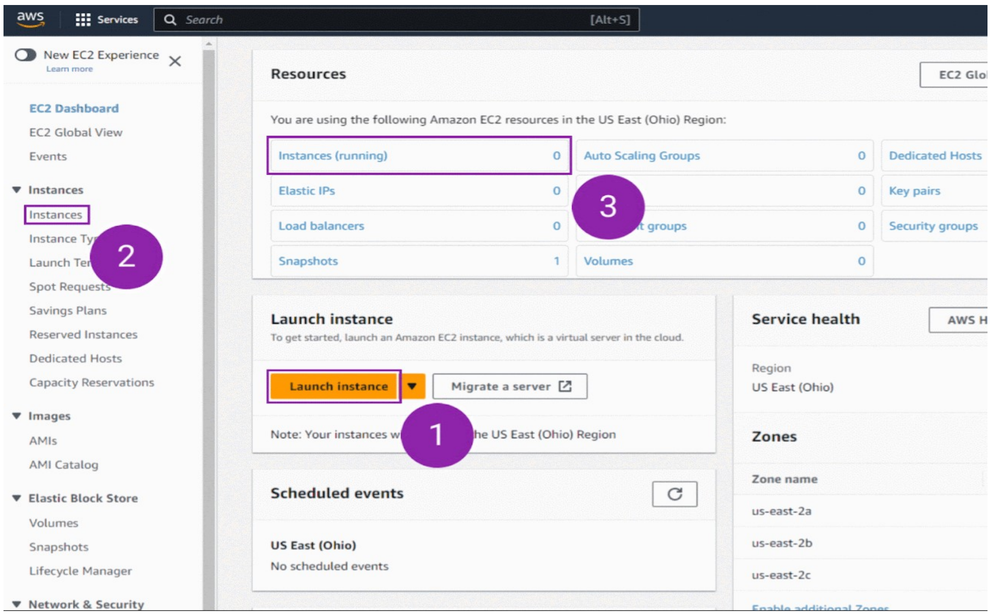


Figure 4.1: AWS Console Window



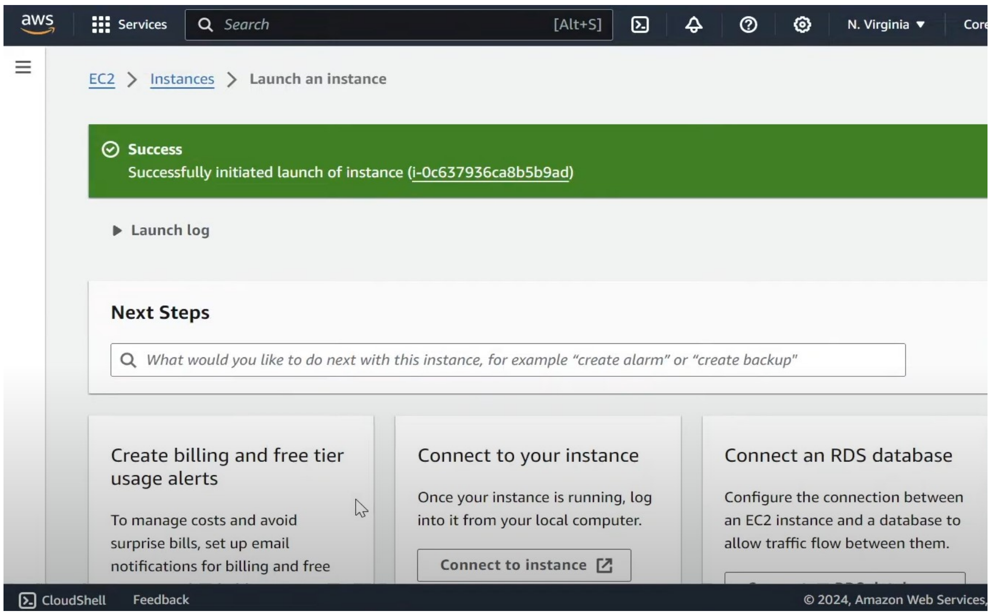
You are using the following Amazon EC2 resources in the US East (Ohio) Region:

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

Note: Your instances w he US East (Ohio) Region

No scheduled events

Figure 4.2: AWS Instance Creation



Success
Successfully initiated launch of instance (i-0c637936ca8b5b9ad)

What would you like to do next with this instance, for example "create alarm" or "create backup"

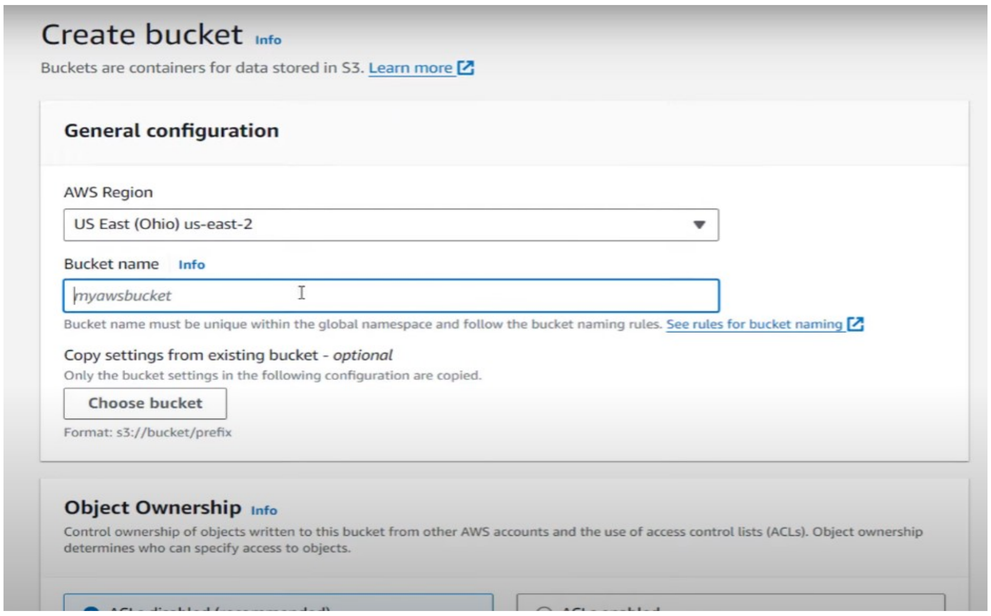
To manage costs and avoid surprise bills, set up email notifications for billing and free

Once your instance is running, log into it from your local computer.

Configure the connection between an EC2 instance and a database to allow traffic flow between them.

© 2024, Amazon Web Services,

Figure 4.3: AWS Instance Launch Success



Create bucket Info

Buckets are containers for data stored in S3. Learn more

General configuration

AWS Region

US East (Ohio) us-east-2

Bucket name Info

myawsbucket

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

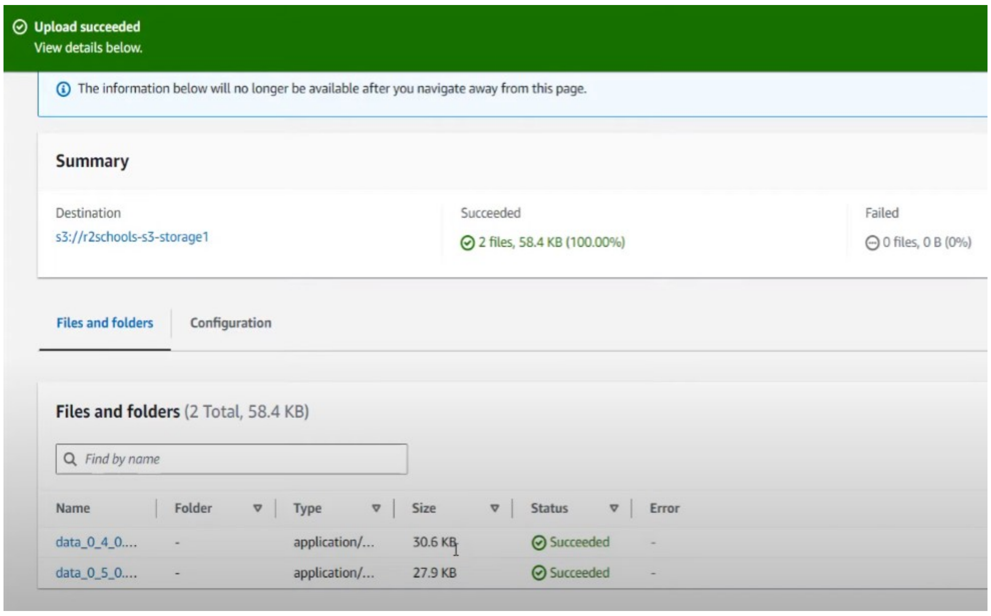
Choose bucket

Format: s3://bucket/prefix

Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

Figure 4.4: S3 Bucket Creation and Launch



Upload succeeded

View details below.

The information below will no longer be available after you navigate away from this page.

Summary

Destination	Succeeded	Failed
s3://r2schools-s3-storage1	2 files, 58.4 KB (100.00%)	0 files, 0 B (0%)

Files and folders | Configuration

Files and folders (2 Total, 58.4 KB)

Find by name

Name	Folder	Type	Size	Status	Error
data_0_4_0....	-	application/...	30.6 KB	Succeeded	-
data_0_5_0....	-	application/...	27.9 KB	Succeeded	-

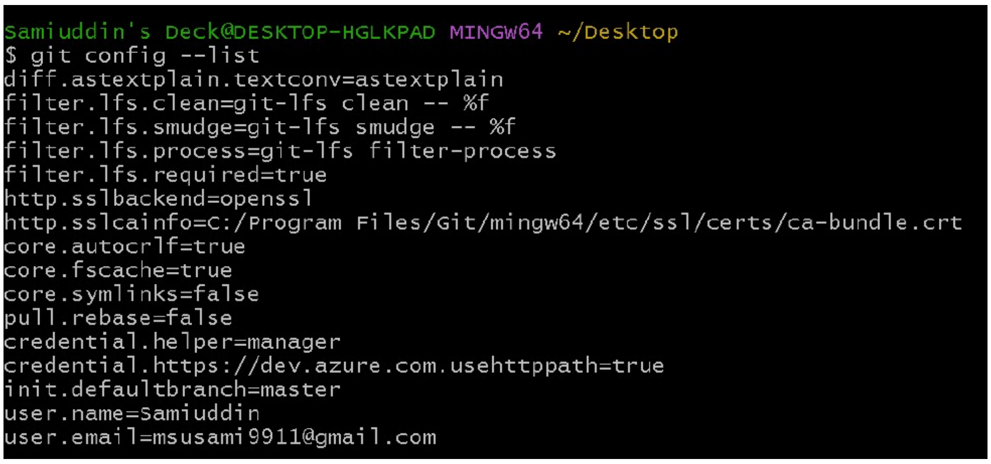
Figure 4.5: S3 Bucket and uploading files

4.2 Assignment 2: Git Commands

4.2.1 Configure Git with Name and Email

To set your username and email, use the following commands:

```
        # Configure your Git username
        git config --global user.name "User_name"
        # Configure your Git email
        git config --global user.email "user@email.com"
```



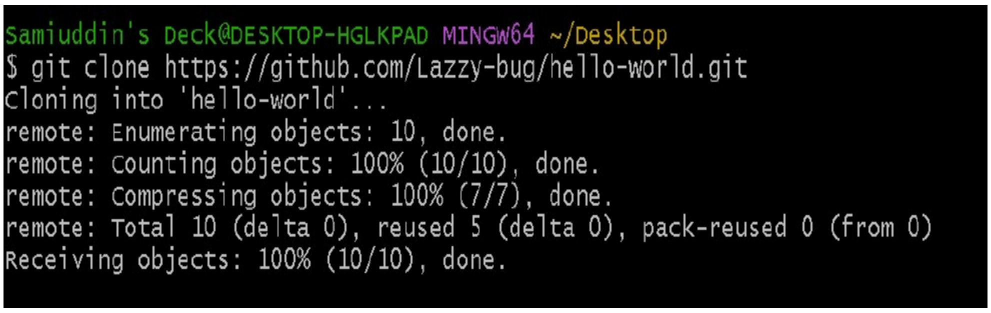
```
Samiuddin's Deck@DESKTOP-HGLKPAD MINGW64 ~/Desktop
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Samiuddin
user.email=msusami9911@gmail.com
```

Figure 4.6: Git config with Name and Email

4.2.2 Clone the Project Repository

To clone the repository from GitHub, use the following command:

```
        git clone https://github.com/Lazzy-bug/hello-world.git
```



```
Samiuddin's Deck@DESKTOP-HGLKPAD MINGW64 ~/Desktop
$ git clone https://github.com/Lazzy-bug/hello-world.git
Cloning into 'hello-world'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 10 (delta 0), reused 5 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (10/10), done.
```

Figure 4.7: Git config with Name and Email

4.2.3 Create a New Feature Branch

To create and switch to a branch called new-feature, use:

```
    git checkout -b new-feature
```

4.2.4 Add a New File Called

To stage the new file for commit, use:

```
    # Create a new file
    feature.py
    # Stage the new file
    git add feature.py
```

4.2.5 Commit Your Changes

To commit your staged changes with a message, use:

```
        git commit -m "Add new feature implementation"
```

4.2.6 Push Your Changes to the Remote Repository

To push your changes in the new-feature branch to the remote repository, use:

```
        git push origin new-feature
```

4.2.7 Pull the Latest Changes from the Main Branch

To pull the latest changes from the main branch of the remote repository, use:

```
        git pull origin main
```

4.2.8 Remove the feature.py File

To remove the feature.py file from your branch, use:

```
    git rm feature.py
```

4.2.9 Commit the Removal and Push Changes

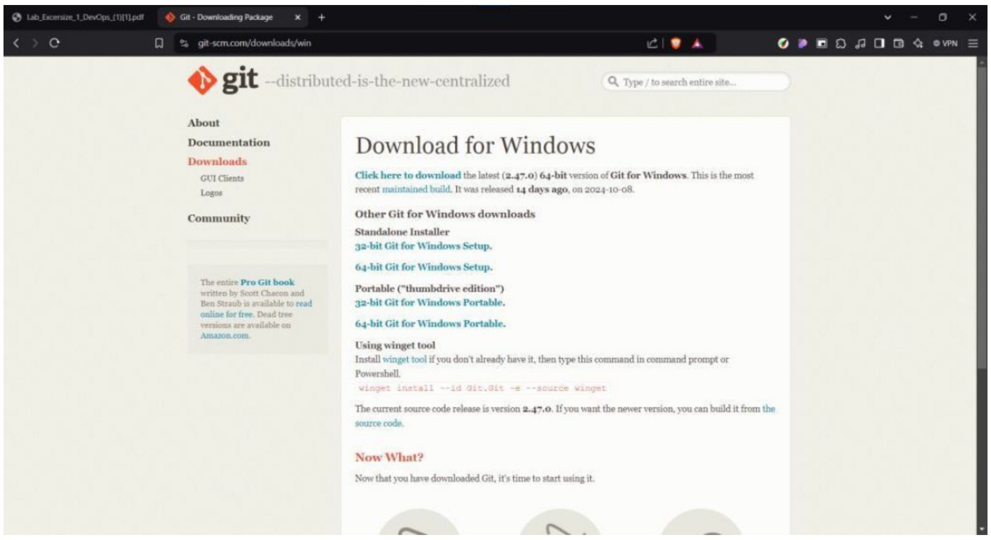
To commit the removal and push the changes to the remote repository, use:

```
    git commit -m "Remove feature.py"
    git push origin new-feature
```

4.3 Assignment 2: Jenkins Job Creation: Implementing Version Controlling Your CI/CD Pipeline

4.3.1 Install Git Bash

Download and install Git Bash from the official website.


Lab_Excersize_1_DevOps_(1)(1).pdf
Git - Downloading Package
git-scm.com/downloads/win
git --distributed-is-the-new-centralized
Type / to search entire site...
About
Documentation
Downloads
GUI Clients
Logos
Community

The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Download for Windows

Click here to download the latest (2.47.0) 64-bit version of Git for Windows. This is the most recent maintained build. It was released 14 days ago, on 2024-10-08.

Other Git for Windows downloads

Standalone Installer
32-bit Git for Windows Setup.

64-bit Git for Windows Setup.

Portable ("thumbdrive edition")
32-bit Git for Windows Portable.

64-bit Git for Windows Portable.

Using winget tool
Install winget tool if you don't already have it, then type this command in command prompt or Powershell.

```
winget install --id Git.Git -e --source winget
```

The current source code release is version 2.47.0. If you want the newer version, you can build it from the source code.

Now What?

Now that you have downloaded Git, it's time to start using it.

Figure 4.8: Git Download Window

4.3.2 Set Up Git Bash

```
gitconfig--globaluser.name"Name"
gitconfig--globaluser.email"name@email.com"
```

4.3.3 Initialize a Git Repository

```
git init
```

4.3.4 Add Files and Commit to Git

```
gitaddfeature.py
gitcommit-m"Addnewfeatureimplementation"
```

4.3.5 Push the Project to GitHub

```
    gitremoteaddoriginhttps://github.com/Lazzy-bug/hello-world.git
    gitpush-u origin master
```

4.3.6 Install Jenkins Plugins

Steps:

- Go to Jenkins Dashboard → ManageJenkins → Manage Plugins.
- Search for and install the required plugins (e.g. Pipeline, Git).

4.3.7 Create a Jenkins Pipeline Job

Steps:

- From the Jenkins dashboard, click on “New Item.”
- Name the job, select “Pipeline” and click “OK.”

4.3.8 Write the Jenkins Pipeline Script

```
    pipeline {
        agentany
        stages {
            stage('Checkout'){
                steps {
                    giturl:'https://github.com/Lazzy-bug/hello-world.git
                        ',
branch:'main'
            }
        }
        stage('Build'){
            steps {
                echo'Buildingtheapplication...'
            }
        }
        stage('Test'){
            steps {
                echo'Runningtests...'
        }
            }
        stage('Deploy'){
            steps {
                echo'Deployingtheapplication...'
                }
            }
}
post{
    success{
        echo'Pipelinesucceeded!'
    }
    failure{
        echo'Pipelinefailed!'
        }
    }
}
```

4.3.9 Run the Jenkins Pipeline

After saving the script, go to the dashboard, select your pipeline job,and click “Build Now.”

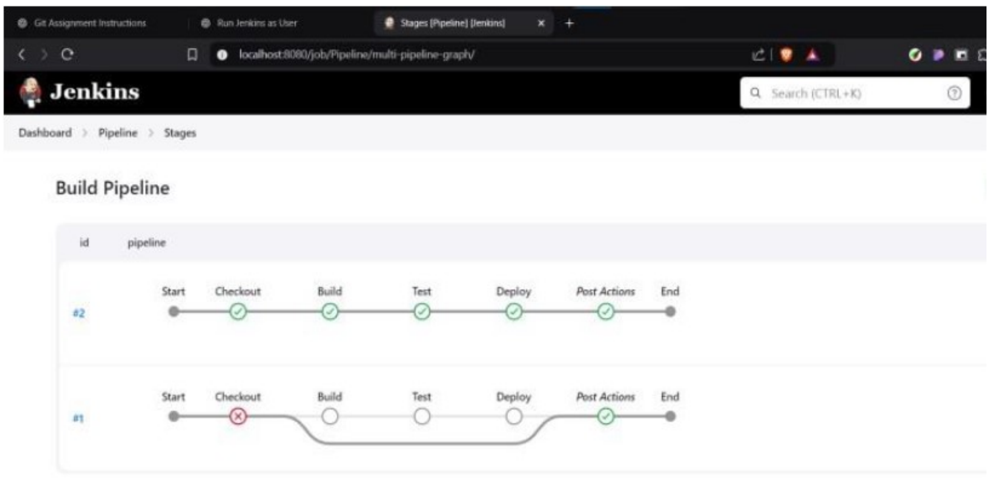
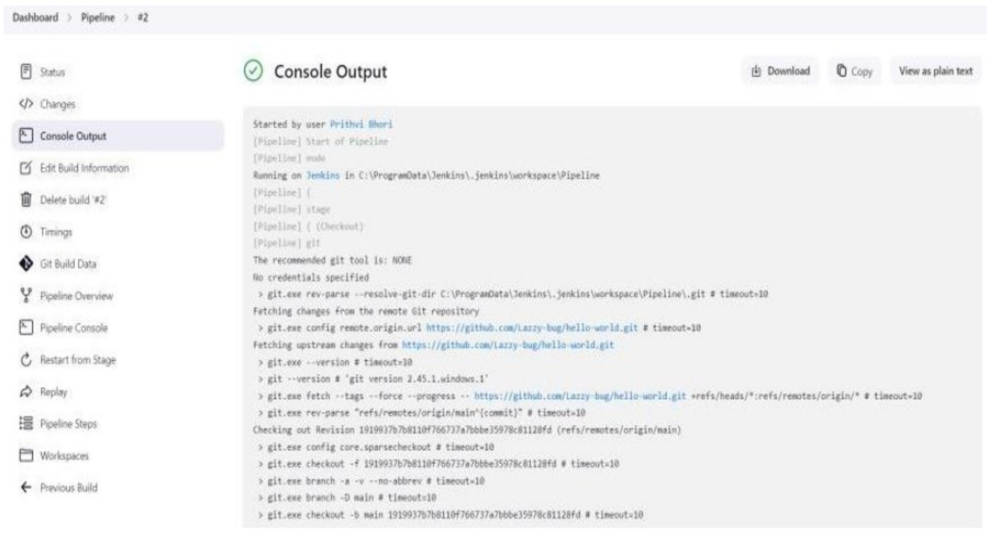


Figure 4.9: Jenkins Pipeline Built Status



Console Output

```
Started by user Prithvi Bhori
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\Pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Checkout)
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
 > git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\.jenkins\workspace\Pipeline\.git # timeout=10
Fetching changes from the remote Git repository
 > git.exe config remote.origin.url https://github.com/Lazzy-bug/hello-world.git # timeout=10
Fetching upstream changes from https://github.com/Lazzy-bug/hello-world.git
 > git.exe --version # timeout=10
 > git --version # 'git version 2.45.1.windows.1'
 > git.exe fetch --tags --force --progress -- https://github.com/Lazzy-bug/hello-world.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git.exe rev-parse "refs/remotes/origin/main^{commit}" # timeout=10
Checking out Revision 1919937b7b8110f766737a7bbbe35978c81128fd (refs/remotes/origin/main)
 > git.exe config core.sparsecheckout # timeout=10
 > git.exe checkout -f 1919937b7b8110f766737a7bbbe35978c81128fd # timeout=10
 > git.exe branch -a -v --no-abbrev # timeout=10
 > git.exe branch -D main # timeout=10
 > git.exe checkout -b main 1919937b7b8110f766737a7bbbe35978c81128fd # timeout=10
```

Figure 4.10: Jenkins Pipeline Console Output

4.4 Assignment 3: Containerizing a Python Application with Docker - Deploying Python in Docker with Flask

4.4.1 Write Your Python Application Code

Create a Python script called app.py with the following content:

```
# app.py
print("Hello from Dockerized Python App by Name!")
```

4.4.2 Create a Dockerfile

Create a Dockerfile in the same directory as app.py with the following content:

```
# Use an official Python runtime as a parent image FROM python:3.8-
    slim
# Set the working directory in the container WORKDIR /app
# Copy the current directory contents into the container at /app
    COPY . /app
# Install any needed packages specified in requirements.txt (skip if
     not needed)
RUN pip install --trusted-host pypi.python.org -r requirements.txt
    || true
# Make port 80 available to the outside world EXPOSE 80
# Define environment variable ENV NAME World
# Run app.py when the container launches CMD ["python", "app.py"]
```

4.4.3 Build the Docker Image

In your terminal, navigate to the directory containing the Dockerfile and run:

```
docker build -t my-python-app
```

4.4.4 Run the Docker Container

Start the container and map port 4000 on your machine to port 80 in the container:

```
docker run -p 4000:80 my-python-app
```

4.4.5 Access Your Python Application

Since app.py currently only prints a message, you'll see the output in the terminal, but there's nothing to view in the browser yet.

4.4.6 Install Flask

Create a requirements.txt file with the following content:

4.4.7 Update app.py

Modify app.py to use Flask and create a simple web server:

```
# app.py
from flask import Flask app = Flask( name )
@app.route("/") def hello():
return "Hello from Flask Dockerized Python App by Samiuddin!"
if name == " main ": app.run(host="0.0.0.0", port=80)
```

4.4.8 Rebuild the Docker Image

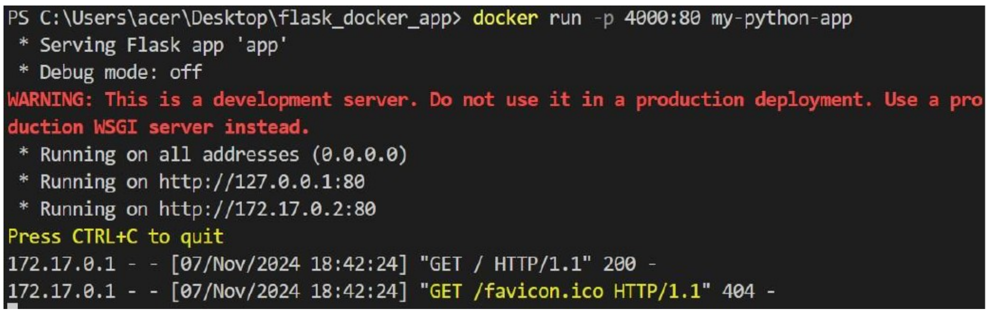
After modifying app.py and creating requirements.txt, rebuild the Docker image:

```
docker build -t my-python-app
```

4.4.9 Run the Container Again

Run the updated container:

```
    docker run -p 4000:80 my-python-app
```

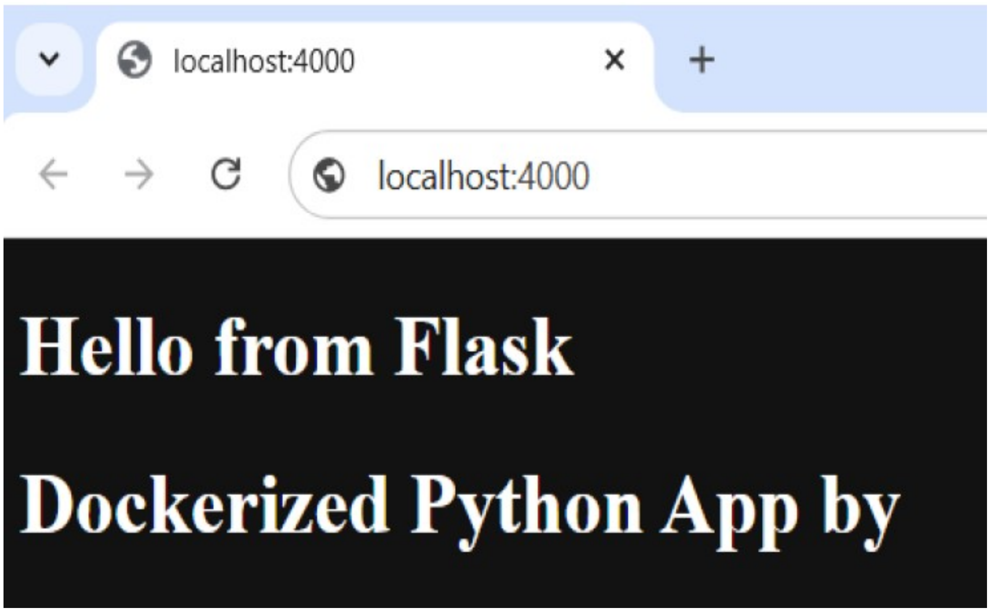


```
PS C:\Users\acer\Desktop\flask_docker_app> docker run -p 4000:80 my-python-app
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a pro
duction WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://172.17.0.2:80
Press CTRL+C to quit
172.17.0.1 - - [07/Nov/2024 18:42:24] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [07/Nov/2024 18:42:24] "GET /favicon.ico HTTP/1.1" 404 -
```

Figure 4.11: Docker Image Running and Live

4.4.10 Access the Flask Application

Open a browser and go to http://localhost:4000. To see ”Hello from Flask Dockerized Python App by Name!”



localhost:4000

localhost:4000

Hello from Flask

Dockerized Python App by

Figure 4.12: Flask Application Output

4.5 Assignment 4: Machine Learning Model Containerized Application Deployment

4.5.1 Build and Train the Machine Learning Model

Prepare the Directory Structure

```
ml-app/
 app/
  model.py
  app.py
  requirements.txt
 Dockerfile
 kubernetes/
 deployment.yaml
 service.yaml
```

4.5.2 Write the Model Code (model.py)

Write the following Python script to train and save the model:

```
import pickle
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier

    def train_model():
        data = load_iris()
        X, y = data.data, data.target
        model = RandomForestClassifier()
    model.fit(X, y)
        with open("model.pkl", "wb") as file:
    pickle.dump(model, file)
    print("Model saved as 'model.pkl'")
    if __name__ == "__main__":
        train_model()
```

4.5.3 Train the Model

Run the script to generate model.pkl:

```
python app/model.py
```

4.5.4 Write the Flask API (app.py)

Create a Flask app to serve the model:

```
python
Copy code
import pickle
from flask import Flask, request, jsonify

app = Flask(__name__)
with open("model.pkl", "rb") as file:
    model = pickle.load(file)

@app.route("/")
def home():
    return "Welcome to ML App by samiuddin!"

@app.route("/predict", methods=["POST"])
def predict():
    data = request.json
    prediction = model.predict([data["features"]])
    return jsonify({"prediction": prediction.tolist()})

if __name__ == "__main__":
app.run(host="0.0.0.0", port=5000)
```

4.5.5 Add Dependencies in requirements.txt

Add the following dependencies:

```
Flask==2.3.3
scikit-learn==1.3.2
```

4.5.6 Containerize the Application Using Docker

Create a Dockerfile with the following content:

```
FROM python:3.9-slim
WORKDIR /app
COPY app/ /app/
COPY model.pkl /app/
RUN pip install -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

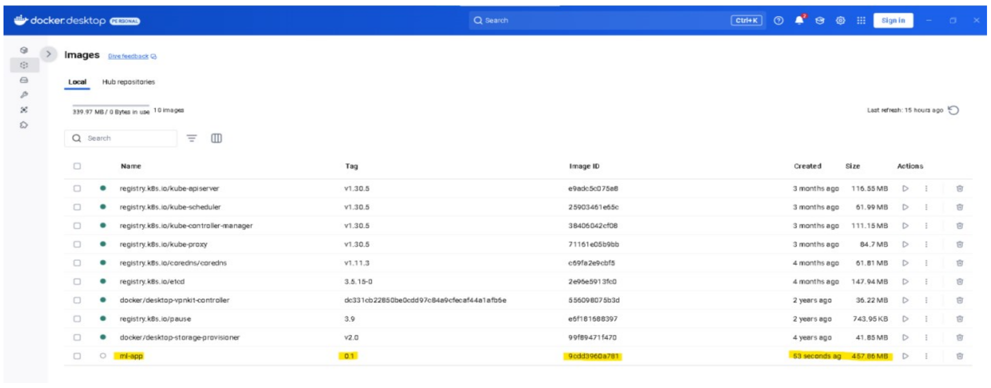
4.5.7 Build the Docker Image

Run the following command to build the image:

```
docker build . -t ml-app:0.1
```

4.5.8 Test the Docker Image

Run the Docker container: Click on run as shown below in docker desktop from the images created.

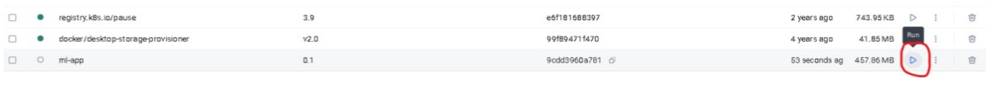


339.97 MB / 0 Bytes in use 10 images

Last refresh: 15 hours ago

Name	Tag	Image ID	Created	Size
registry.k8s.io/kube-apiserver	v1.30.5	e9adc5c075a8	3 months ago	116.55 MB
registry.k8s.io/kube-scheduler	v1.30.5	25903461e65c	3 months ago	61.99 MB
registry.k8s.io/kube-controller-manager	v1.30.5	38406042cf08	3 months ago	111.15 MB
registry.k8s.io/kube-proxy	v1.30.5	71161e05b9bb	3 months ago	84.7 MB
registry.k8s.io/coredns/coredns	v1.11.3	c69fa2e9cbf5	4 months ago	61.81 MB
registry.k8s.io/etcd	3.5.15-0	2e96e5913fc0	4 months ago	147.94 MB
docker/desktop-vpnkit-controller	dc331cb22850be0cdd97c84a9cfecaf44a1afb6e	556098075b3d	2 years ago	36.22 MB
registry.k8s.io/pause	3.9	e6f181688397	2 years ago	743.95 KB
docker/desktop-storage-provisioner	v2.0	99f89471f470	4 years ago	41.85 MB
ml-app	0.1	9cdd3960a781	53 seconds ag	457.86 MB

Figure 4.13: Docker Desktop Images



registry.k8s.io/pause	3.9	e6f181688397	2 years ago	743.95 KB
docker/desktop-storage-provisioner	v2.0	99f89471f470	4 years ago	41.85 MB
ml-app	0.1	9cdd3960a781	53 seconds ag	457.86 MB

Figure 4.14: Docker Image Generated

4.5.9 Deploy the Application Using Kubernetes

Start Minikube: Enable kubernetes and initialize a local Kubernetes cluster:

```
minikube start
```

4.5.10 Ensure Minikube is running:

```
kubectl cluster-info
```

4.5.11 Create Kubernetes Manifests

1. Write deployment.yaml:

```
            apiVersion: apps/v1
            kind: Deployment
            metadata:
                name: ml-app
                spec:
                replicas: 1
                selector:
            matchLabels:
            app: ml-app
                template:
                metadata:
                 labels:
            app: ml-app
                spec:
                containers:
                - name: ml-app
                image: lazzyxbug/ml-app:latest
                ports:
        - containerPort: 5000
```

2. Write service.yaml:

```
            apiVersion: v1
            kind: Service
            metadata:
                name: ml-app-service
            spec:
                selector:
                app: ml-app
            ports:
            - protocol: TCP
            port: 80
            targetPort: 5000
            type: NodePort
```

3. Apply the Kubernetes Manifests

```
        kubectl apply -f kubernetes/deployment.yaml
    kubectl apply -f kubernetes/service.yaml
```

4. Check the status of Pods and Services:

```
        kubectl get deployments
        kubectl get pods
        kubectl get svc
```

5. Use Minikube to access the service:

```
minikube service ml-app-service
```

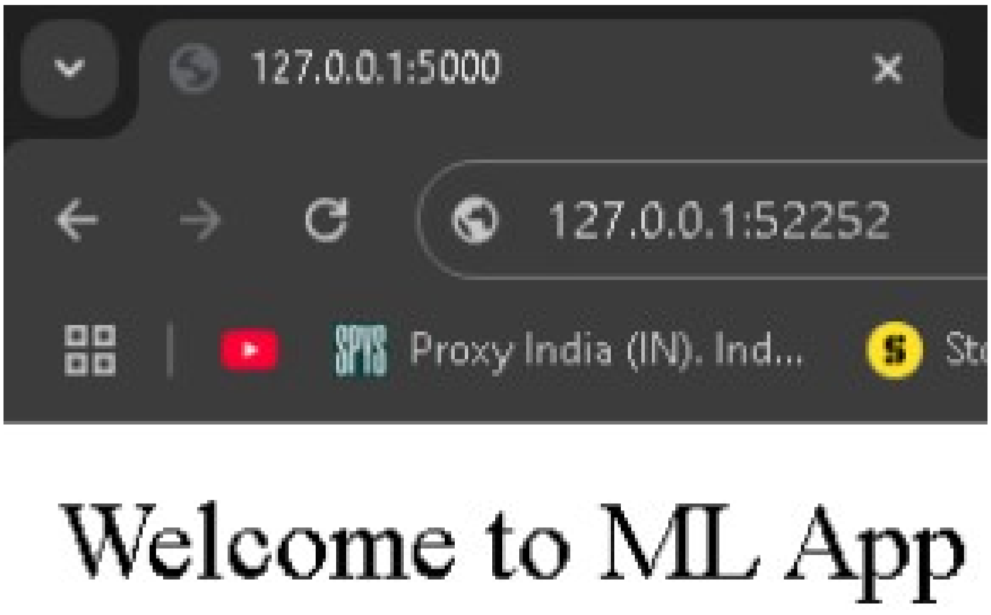
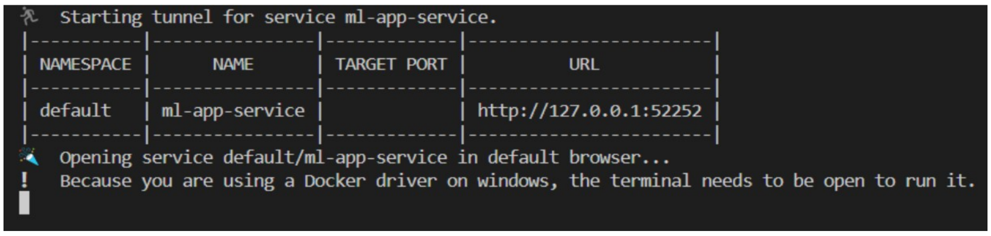


Figure 4.15: Minikube Output



```
🏃  Starting tunnel for service ml-app-service.
|-----------|----------------|-------------|------------------------|
| NAMESPACE |      NAME      | TARGET PORT |          URL           |
|-----------|----------------|-------------|------------------------|
| default   | ml-app-service |             | http://127.0.0.1:52252 |
|-----------|----------------|-------------|------------------------|
🎉  Opening service default/ml-app-service in default browser...
❗  Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

Figure 4.16: Minikube Service Running

CHAPTER 5

PROJECT

5.1 Weather Alert App: Scalable Deployment On IBM Cloud Kubernetes

5.1.1 Phase 1- Problem Analysis

5.1.1.1 Identifying the Problem and Core Challenges

Problem Statement: The current process for deploying the weather alert app faces several roadblocks:

1. **Manual Deployments:** Deployments are handled manually, causing delays, human errors, and interruptions in the system's performance. This leads to significant challenges in providing real-time weather alerts.

2. **No Automation:** Without a proper CI/CD pipeline, processes like testing, building, and deploying take longer, increasing the risk of bugs or failures reaching production, which can result in missed or delayed alerts.

3. **Inconsistent Environments:** Configurations for development, staging, and production environments differ, causing unexpected failures when moving code between them, especially in critical emergency situations.

4. **User Expectations:** Users rely on the app for real-time weather alerts, and they expect immediate notifications of severe weather changes. The current deployment process slows down our ability to roll out new features or fixes, affecting the timeliness and accuracy of alerts.

Key areas to focus on:

- **Deployment Issues:** Manual workflows cause errors and delays, directly impacting the timeliness of emergency weather notifications.

- **CI/CD Challenges:** Lack of automation causes slow updates, introducing potential risks, especially in critical weather alert systems.

- **User Needs:** Users need fast, reliable, and real-time emergency weather alerts, especially when severe weather events are imminent.

5.1.1.2 Application Needs and the Right Tools

What the App Needs to Do (Functional Requirements):

1. **Deliver Real-Time Weather Updates:** Provide users with accurate, real-time forecasts and alerts.
2. **User Accounts:** Allow users to create accounts, save favorite locations, and personalize settings like notifications.
3. **Integrate Live Data:** Fetch and process weather data from APIs and display it in an easy-to-understand format.
4. **Send Notifications:** Alert users about severe weather changes or updates promptly.
5. **Visualize Data:** Present weather trends and reports visually for better user understanding.

How the App Should Perform (Non-Functional Requirements):

1. **Fast Performance:** The app must load quickly, provide real-time updates, and offer instant alerts, even when handling a high volume of users and requests.
2. **Scalability:** The system should be able to scale with an increasing user base and more frequent weather data updates as the application grows.
3. **Security:** Ensure user data is encrypted, and all communications are secure.
4. **Reliability:** The app must be available 24/7, especially during weather events. Quick recovery mechanisms should be in place for any service failures.
5. **Ease of Maintenance:** Write clean, maintainable code and automate repetitive tasks to simplify updates.

5.1.1.3 Tools and Technologies

1. **Development Stack**
    - **Backend:** Python (Flask or Django) for processing requests and managing business logic.
    - **Frontend:** React for a smooth, user-friendly interface.
    - **Database:** PostgreSQL to store user data and weather forecasts.
2. **Version Control**
    - **Git and GitHub:** For tracking changes, collaborating, and managing the codebase.

3. **CI/CD Pipeline Tools**

   (a) **Automation and Orchestration:**

   - **Jenkins or GitLab CI:** To automate builds, tests, and deployments.
   - **Docker:** To package the app for consistent deployment.
   - **Kubernetes:** To manage containers and scale the app efficiently.

   (b) **Infrastructure as Code:**

   - **Terraform:** To automate cloud resource provisioning

5.1.2 Solution Architecture:

To streamline the deployment process for the Node.js Weather Alert application, the solution architecture will integrate multiple DevOps tools like GitHub, Jenkins, Docker, and Kubernetes. We will automate code commits and establish a CI/CD pipeline for seamless deployments.

Directory Structure:

```
weather-alert-app
 backend/
  server.js
  requirements.txt
 frontend/
  app.js
  index.html
  script.js
  styles.css
 Dockerfile
 kubernet/
  deployment.yaml
  service.yml
 README.md
```

5.1.2.1 Project Setup

Node.js Application Initialization

1. **Create the main project folder:**

```
bash
mkdir weather-alert-app && cd weather-alert-app
```

2. **Initialize a Node.js project:**

```
bash
npm init -y
```

3. **Create an index.js file:**

```
// filepath: /weather-alert-app/backend/server.js
const express = require('express');
const app = express();
app.get('/', (req, res) => {
res.send('Weather Alert App');
});
app.listen(5000, () => {
console.log('Server started on port 5000');
});
```

4. **Create a requirements.txt file:**

```
Flask
Flask-SQLAlchemy
requests
flask-cors
```

5. **Create a Dockerfile for the backend:**

```
# filepath: /c:/Users/hp/OneDrive/Desktop/weather-alert-app/backend
    /Dockerfile
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt /app/requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
COPY . /app
EXPOSE 5000
CMD ["python", "server.js"]
```

6. **Create a Dockerfile for the frontend:**

```
# filepath: /c:/Users/hp/OneDrive/Desktop/weather-alert-app/
    frontend/Dockerfile
FROM node:14-slim
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "app.js"]
```

5.1.2.2 Version Control Setup

GitHub Repository Creation

1. **Initialize a Git repository:**

```
bash
git init
```

2. **Create a .gitignore file:**

```
bash
echo node_modules/ > .gitignore
echo .env >> .gitignore
```

3. **Add and commit the initial codebase:**

```
bash
git add .
git commit -m "Initial commit of dev-ops structure"
```

4. **Create a GitHub repository (via the GitHub interface) and push the local repository:**

```
bash
git remote add origin <repository_url>
git push -u origin master
```

5.1.2.3 CI/CD Pipeline Design And Implementation

Jenkins Setup

1. Install Jenkins and configure required plugins (Git, Docker, Kubernetes, etc.).
2. Create a Jenkins job triggered by code changes in the GitHub repository.

Jenkinsfile Configuration

The Jenkinsfile defines the pipeline stages:

1. **Checkout:** Pull the latest code from GitHub.
2. **Build:** Build the Node.js application.
3. **Test:** Run automated tests.
4. **Docker Image Creation:** Build and tag the Docker image.
5. **Push to Container Registry:** Push the Docker image to a central repository.
6. **Deploy:** Deploy the Docker container to Kubernetes.

5.1.3 Solution Development And Testing

5.1.3.1 Setting Up Multi-Cloud Environment and Configuring Necessary Tools

Create Accounts on Multi-Cloud Platforms

1. **IBM Cloud:**
   - Navigate to IBM Cloud .
   - Sign up for an account (or log in if you already have one).
   - Ensure you have a billing account set up to access IBM Cloud services.
2. **AWS (Amazon Web Services):**
   - Navigate to AWS .
   - Sign up for an account (or log in if you already have one).
   - Set up billing and enable access to AWS services like EKS (Elastic Kubernetes Service) and ECR (Elastic Container Registry).

5.1.3.2 Implementing Containerization and Pushing to Multi-Cloud Container Registries

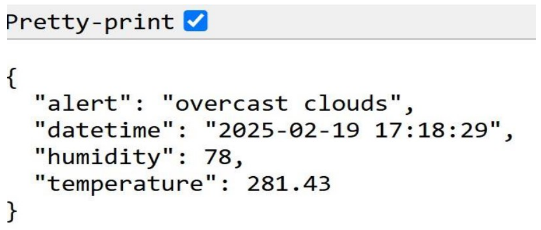
Step 1: Clone the Weather Alert App Repository

1. Clone the Weather Alert App from GitHub:

Step 2: Dockerize the Weather Alert App

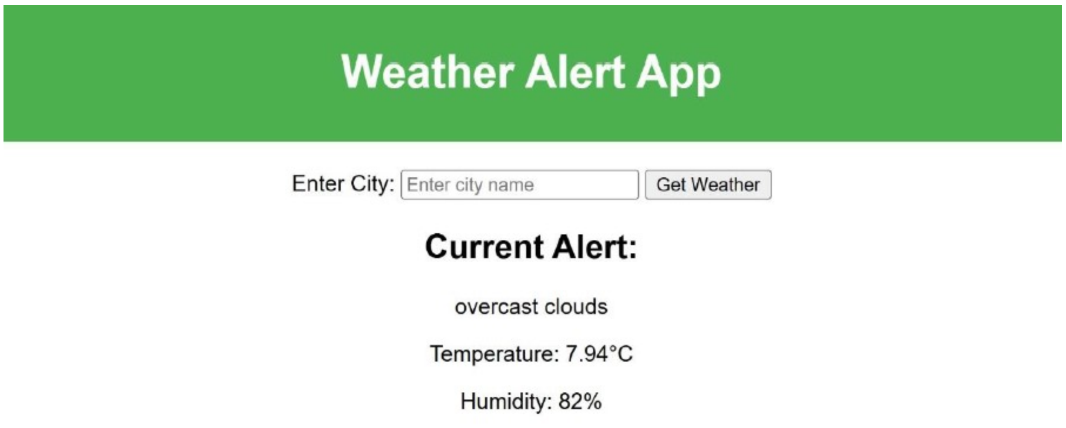
1. **Create Dockerfile:** Create a Dockerfile in the root directory of the Weather Alert App.
2. **Build Docker Image:** Build the Docker image for the Weather Alert App:

```
docker-compose up --build
```


Pretty-print ✓

```
{
  "alert": "overcast clouds",
  "datetime": "2025-02-19 17:18:29",
  "humidity": 78,
  "temperature": 281.43
}
```

Figure 5.1: Back End View



Weather Alert App

Enter City: Enter city name Get Weather

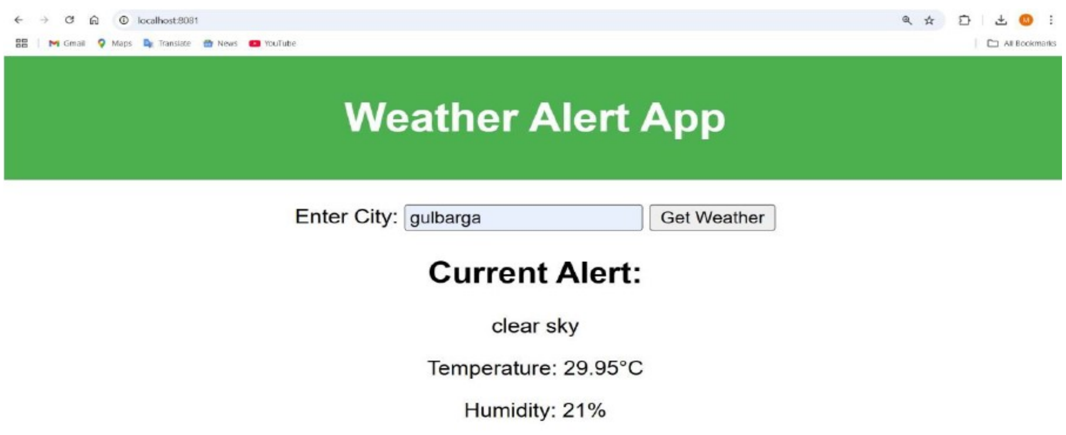
Current Alert:

overcast clouds

Temperature: 7.94°C

Humidity: 82%

Figure 5.2: Home Page



Weather Alert App

Enter City: gulbarga Get Weather

Current Alert:

clear sky

Temperature: 29.95°C

Humidity: 21%

Figure 5.3: Enter city name to get the weather

Step 3: Push Docker Images to Multi-Cloud Container Registries

Push to IBM Cloud Container Registry

- **Tag the image:** docker tag weather-alert-app:1.0 us-
- south.icr.io/weather-alert-ns/weather-alert-app:1.0
- **Log in to IBM Cloud Container Registry :** ibmcloud cr login
- **Push the image:** docker push us-south.icr.io/weather-alert-ns/weather-alert-app:1.0

Similarly, Push to AWS Elastic Container Registry (ECR)

5.1.3.3 Testing The Solution

Step 1: Set Up Minikube and Deploy Flask Blog App

1. **Start Minikube:**

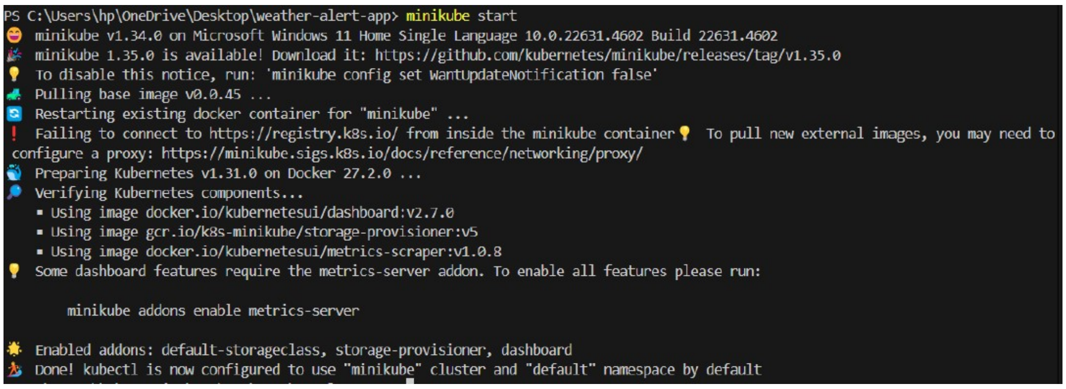
```
minikube start
kubectl cluster-info
```

2. **Apply YAML Files:**

```
kubectl apply -f
deployment.yaml
kubectl apply -f kubernetes/service.yaml
```

3. **Load the Docker image into Minikube:**

```
minikube image load weather-alert-app:latest
```

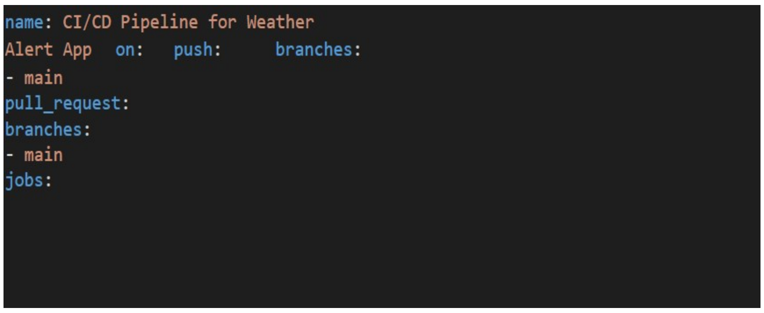


```
PS C:\Users\hp\OneDrive\Desktop\weather-alert-app> minikube start
😄  minikube v1.34.0 on Microsoft Windows 11 Home Single Language 10.0.22631.4602 Build 22631.4602
🎉  minikube 1.35.0 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.35.0
💡  To disable this notice, run: 'minikube config set WantUpdateNotification false'
🚜  Pulling base image v0.0.45 ...
🔄  Restarting existing docker container for "minikube" ...
❗  Failing to connect to https://registry.k8s.io/ from inside the minikube container💡  To pull new external images, you may need to
 configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
🐳  Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
🔎  Verifying Kubernetes components...
    ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
💡  Some dashboard features require the metrics-server addon. To enable all features please run:

        minikube addons enable metrics-server

🌟  Enabled addons: default-storageclass, storage-provisioner, dashboard
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Figure 5.4: Starting Minikube

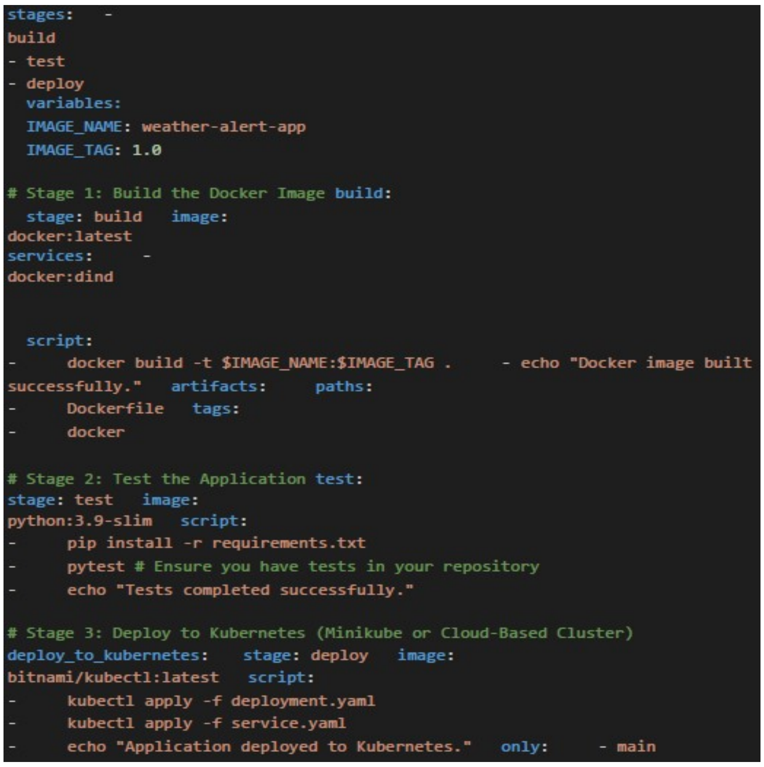
Step 2: Testing CI/CD Integration

1. **Set Up GitHub Actions:**
   - Create a .github/workflows/ci-cd.yaml file in the repository.
   - Automate build, test, and deployment stages using GitHub Actions.
2. **Example GitHub Actions Workflow:**
   - **Screenshot:** Include a screenshot of the GitHub Actions workflow running successfully in the GitHub repository: .github/workflows/deploy.yml



```
name: CI/CD Pipeline for Weather
Alert App   on:   push:     branches:
- main
pull_request:
branches:
- main
jobs:
```

Figure 5.5: GitHub Actions workflow



```
stages:   -
build
- test
- deploy
  variables:
  IMAGE_NAME: weather-alert-app
  IMAGE_TAG: 1.0

# Stage 1: Build the Docker Image build:
  stage: build   image:
docker:latest
services:     -
docker:dind


  script:
-     docker build -t $IMAGE_NAME:$IMAGE_TAG .     - echo "Docker image built
successfully."   artifacts:     paths:
-     Dockerfile   tags:
-     docker

# Stage 2: Test the Application test:
stage: test   image:
python:3.9-slim   script:
-     pip install -r requirements.txt
-     pytest # Ensure you have tests in your repository
-     echo "Tests completed successfully."

# Stage 3: Deploy to Kubernetes (Minikube or Cloud-Based Cluster)
deploy_to_kubernetes:   stage: deploy   image:
bitnami/kubectl:latest   script:
-     kubectl apply -f deployment.yaml
-     kubectl apply -f service.yaml
-     echo "Application deployed to Kubernetes."   only:     - main
```

Figure 5.6: Build, test, and deployment stages using GitHub Actions

5.1.3.4 Future Improvements

1. **Enable Autoscaling:**
   - Configure Horizontal Pod Autoscaler (HPA) for Kubernetes to automatically scale the Weather Alert App based on CPU or memory usage.
2. **Integrate Advanced CI/CD Pipelines:**
   - Use tools like Jenkins or Tekton Pipelines for more advanced CI/CD workflows.

3. **Add Automated Vulnerability Scanning:**

    - Integrate automated vulnerability scanning during the CI/CD process using tools like IBM Cloud's built-in scanning or third-party solutions like Anchore or Trivy.

4. **Expand Multi-Cloud Strategy:**

    - Explore additional cloud providers like Google Cloud Platform (GCP) or Azure for further diversification.

CHAPTER 6

REFLECTION

The internship from Rooman Technologies was very useful. The training was effective and taught us the efficient ways to build solutions to the problem statements within a given time. The best part about internships was that relevant experience and knowledge about various technologies. The trainers had a different way of teaching where they made sure that we understood the concepts by giving us time to practise the concepts practically. The effective training throughout the internship helped us to take the assessment without any difficulties.

The idea of internship program sees merit in attempting to shorten the period on training that is often significant duration to orient the trainee or newly inducted person onto the project. The internship covered the concepts of python, AI models, databases, docker, CI/CD pipelines, cloud computing with IBM and AWS and how to use these tools in various assignments and projects.

In this DevOps internship, we learnt the basics like Git Bash, Jenkins, Virtualization and Containerization. We also got an opportunity to know about Kubernetes, Git Lab, NodeJs app deployment and in depth about docker images and orchestration. We also did six to seven assignments and a project on DevOps.

The internship session has been a great learning journey helping the participants in the internship program to understand the concepts of DevOps Engineering. It also helped us improve our logical thinking. It helped us to improve our communication skills. They taught us to manage the time so that we could code maximum in limited or specified time. We realised that soft skills contribute to a positive work environment and help us maintain an efficient workflow.

CHAPTER 7

CONCLUSION

The need for internship is more emergent as one need to be oriented with self-learning capacities required at a very short notice in industry. Got to know the industrial standards, the skills that are required to help us contribute and grow with the industry, the ability of learning by our own, logical thinking and many more. The internship program has made me understand the career opportunities that can be explored. An understanding is grown about the lifelong learning one need to sustain in the industry and the curious and openminded attitude one need to have for the same.

The internship helped in improving the interpersonal skills. In this internship, we learnt how to perform continuous integration and development by means of tools and applications. The trainer also helped us in improving our logical thinking by giving us many assignments to write our own codes. Some of the technical outcomes of the training were that we could logically examine and provide considerable solutions.

REFERENCES

1. Amazon Web Services, Inc. (2024). Amazon EC2, S3, CodePipeline, and CodeDeploy documentation. Retrieved from https://aws.amazon.com/documentation/

2. Docker Inc. (2024). Docker documentation. Retrieved from https://docs.docker.com/

3. Flask Pallets Project. (2024). Flask documentation. Retrieved from https://flask.pallets projects.com/

4. GitHub, Inc. (2024). GitHub and Codespaces documentation. Retrieved from https://docs. github.com/

5. Google Cloud. (2024). Google Kubernetes Engine (GKE) documentation. Retrieved from https://cloud.google.com/kubernetes-engine

6. IBM Corporation. (2024). IBM Cloud DevOps Services documentation. Retrieved from https://cloud.ibm.com/docs/devops?topic=devops-getting-started

7. Jenkins Project. (2024). Jenkins User Documentation. Retrieved from https://www.jenkins. io/doc/

8. Wadhwani Foundation. (2024). Entrepreneurship and Skilling Program Overview. Retrieved from https://wadhwanifoundation.org/

9. Python Software Foundation. (2024). Python 3.12 documentation. Retrieved from https://docs.python.org/3/

10. Microsoft. (2024). Azure Kubernetes Service (AKS) documentation. Retrieved from https://learn.microsoft.com/en-us/azure/aks/