**Unlocking the Future of Cybersecurity: Pioneering with DevSecOps and AI-Driven Chatbots**

**Aishwarya Suryakant Waghmare**

School of Computer Science and Information Technology

Symbiosis Skills and Professional University

Pune, India

[aishwaryawaghmare15@gmail.com](mailto:aishwaryawaghmare15@gmail.com)

Animish Basavaraj Pavate

Information Systems Manager

Information Technology Capgemini India

[animish77@gmail.com](mailto:animish77@gmail.com)

**ABSTRACT**

The DevSecOps approach offers a strategic method for developing and incrementally delivering architectural solutions through the applications of agile and scrum frameworks. It plays a pivotal role in project success by integrating security measures into the development process. This paper investigates the complexities of developing a chatbot capable of notifying engineers about critical key performance indicators (KPIs) within a DevSecOps framework. Additionally, it explores the process of designing a user interface for monitoring customer KPIs and constructing an analytics dashboard using diverse analytics libraries. The study uses natural language processing and advanced deep learning techniques for KPI monitoring to discern customer intentions and filter out irrelevant log data. While previous inquiries have examined the application of MLOps and AI in chatbots, the need for further exploration in this domain persists. This paper examines the practical applications of DevSecOps and deep learning in the nascent stages of chatbot development, employing a variety of programming languages and web development strategies.

**Keywords –** DevSecOps, DevOps, MLOps, Deep learning, Cybersecurity, Key Performance Indicators, Artificial Intelligence, Natural Language Processing, Chatbot, Agile Software Model.

1. **INTRODUCTION**

Artificial intelligence is being used in many domains, including development and operations, to put intelligence into the system. DevSecOps is a method that combines security practices into the DevOps process, considering incorporating security early and continuously throughout the software development lifecycle (SDLC). It emphasizes collaboration, automation, and a shared responsibility model among development, operations, and security teams to deliver secure software efficiently and effectively. Many models are considered while developing a system that applies various phases, from planning to maintenance.

The Waterfall Model, a traditional approach in software development, continues through linear and sequential methods as shown in figure 1. Its simplicity, cost-effectiveness, and straightforward testing processes despite the model limits in handling complex project demands [1]. It embodies a considerable risk and uncertainty factor due to its rigidity, making alterations during any stage of development a challenging endeavor, thereby evaluating the project's trajectory only at its conclusion [2]. These limitations concentrated on use of new model that supports flexibility and prioritizes the rapid delivery.

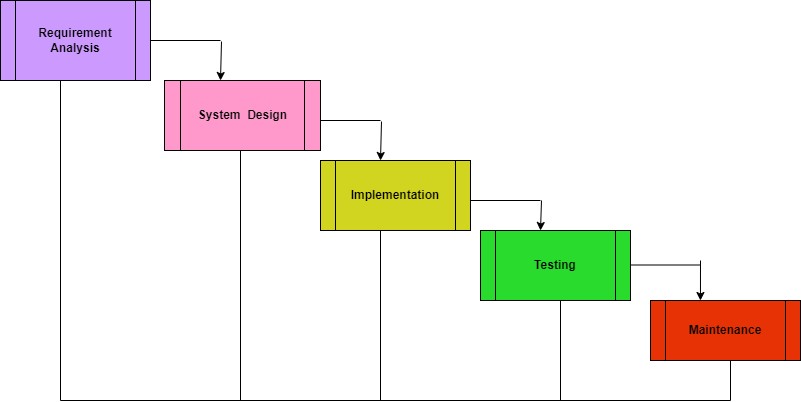


Figure 1: Work Plan in Waterfall Approach

Now a day’s companies are stressing on agile process. The Agile Software Model emphasizes flexibility and prioritizes the satisfaction of customers through the rapid delivery of operational software. This method proves to be cost-efficient by enabling early detection of errors, thereby ensuring the production of superior-quality software. It is characterized by its minimal resource requirement, and ease of management, and is exceptionally well-suited for expansive and enduring projects that need direct interaction among stakeholders. [2] Nonetheless, it faces several challenges, such as insufficient documentation, the potential for prolonged timelines in intricate projects, and the inherent risks of its continuously evolving nature. Furthermore, the model's emphasis on meeting customer needs may hinder the accurate prediction of the project's outcomes. The working plan in agile software model is represented in figure 2. The task that has to be planned and undertaken for the next 15 days( depends on company strategy like maybe 30 days) listed in the product backlog. The sprint planning filters the tasks that is to be possible to complete. The sprint backlog prioritizes the tasks. The scrum team review and prioritize the tasks. The scrum team works for starts executing the task and provides daily updates on progress and blockers if any. At the end each task is reviewed for its success as per deliverable and if there are any blockers then that move to product backlog for completing at later stage. In the spring retrospective discussion, what went well in the sprint and what could have done better is discussed.

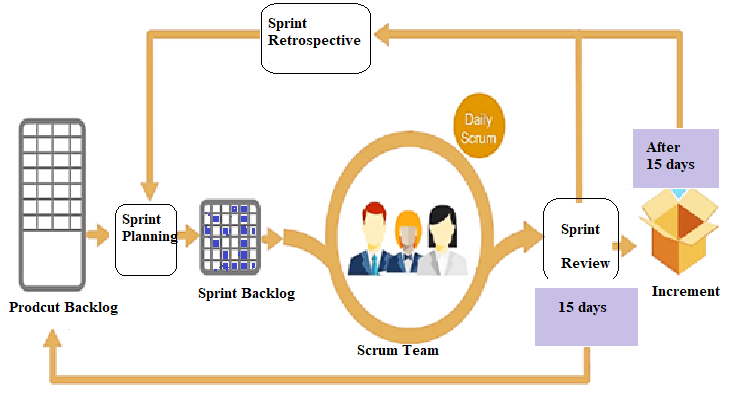


Figure 2: Work Plan in Agile Software Model

Agile and DevOps strategies are focused on meeting customer needs through the prompt and ongoing delivery of valuable software solutions. These methodologies emphasize frequent releases of functional software, favoring shorter development cycles [4]. By automating tasks that do not require human intervention, Agile and DevOps enhance team collaboration and ease the creation of software that can be readily shipped. This is conducted by utilizing sophisticated automation technologies to optimize every stage of the software development process. The benefits of DevSecOps are represented in figure 3. This gives the relationship between agile process and Devops.

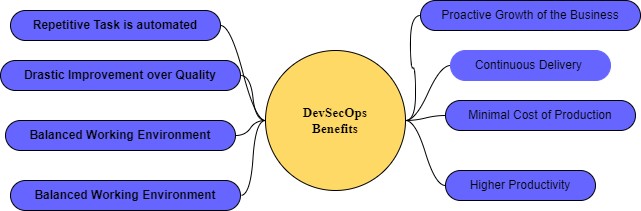


Figure 3: Benefits of DevSecOps

The growing interest in DevSecOps is attributable to its promising capacity to refine software development practices. A deep dive into DevSecOps, through hands-on experience, is crucial for fully grasping its methodologies and their execution [5]. This investigation shows that DevSecOps significantly improves teamwork and dialogue among the various groups involved in software creation, thereby accelerating the release of more robust software products. The discussion extends to the exploration of essential DevSecOps tools and technologies, including Continuous Integration (CI) and Continuous Deployment (CD).

Future studies could aim to unearth optimal strategies for DevSecOps application across a spectrum of business types and sectors [4]. Moreover, it's critical to evaluate how DevSecOps influences software security and its effectiveness in crafting software for cutting-edge technologies such as artificial intelligence and machine learning.

This work highlights the critical role of direct DevSecOps experience for development teams, particularly those leveraging cybersecurity methods [6]. It sheds light on the advantages and obstacles of embracing DevSecOps in the context of developing a chatbot for start-ups, using a variety of programming languages and underlines the pressing need for continued exploration in this evolving domain. The other section of the article discusses the methodology and cases.

1. **METHODOLOGY**

This work first concentrated on the different scenarios such as infrastructure automation using terraform and building a Docker Jenkins Pipeline for CI/CD Workflow

* 1. **Case 1: Infrastructure Automation via Terraform**

This work aimed at streamlining infrastructure setup through Terraform automation, along with integrating necessary automation tools within the setup.

**Tools used:** Terraform, AWS with necessary security credentials, and Keypair were the primary tools employed for this endeavor.

**Procedure:** The process involved initiating an EC2 instance via Terraform, followed by the installation of Jenkins, Java, and Python on the instance [8].

**Methodology:** Leveraging DevSecOps methodologies, the experiment optimized infrastructure provisioning by deploying a centralized Jenkins server. Terraform facilitated the provisioning of infrastructure components, whereas Ansible was used for configuration management and application deployment, enhancing provisioning efficiency and reducing setup time and costs. This approach allowed for the creation of virtual machines through Terraform, with Ansible deploying all necessary applications on these machines, resulting in a quicker and more reliable infrastructure deployment [7].

**Source Code:** A sample source code snippet for setting up Terraform and initiating the provisioning process was provided, illustrating the practical steps taken in the experiment [8].

**$ wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg**

**$ echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list$ sudo apt update && sudo apt install terraform**

**$ terraform -version**

**$ terraform**

**$ aws –version**

**$ sudo vi .aws/credentials**

[awsterraform]

aws\_access\_key\_id = "AKIAYHYCC4VECXH6OVGC"

aws\_secret\_access\_key = "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"

$ sudo ssh-keygen -t rsa -b 2048

Generating public/private rsa key pair.

Enter file in which to save the key (/root/.ssh/id\_rsa): /home/aishwaryawaghma/key/tfkey

ssh-rsa [7] AAAAB3NzaC1yc2EAAAADAQABAAABAQDOkZjyYKt19Om7cl6xkwyB5bIDt3Qj+AQFyp1ilee1Akk3Gca2pZRxGrMKh2iQMlwNf5xwmMqUY+rXRxEM0iC68SXM76JfrHXtj/ImPpdxymEGV5i4I+Ch6BlqxyHs2lC8DlgBp8byamyJq0Td3TnRlYFwLZWwZZHC5iQEVFcOtZUk2b8e9xGw0+p5iJP5DkvrlQEY4bsypw5JEspe1Ead+0aiUIdBtB

4SBwb5odwUwqYh508SoDScQxzjGeKub6G912ynMBQ+hgS0xLvhg0cWF0nKo9wqu7yGTa+20x2TicKVODa2S6FdsPxyT8cZwpLkns0r2RRjUGnyoitRDG2n root@ip-172-31-42-5

**$ sudo nano instance.tf**

provider "aws" {

region = "us-east-1"

access\_key = "AKIAYHYCC4VECXH6OVGC"

secret\_key = "FT3PUOLVrUgleQi5tqLHKDx7CJdQ2gn/813KqDdu"

}

resource "aws\_instance" "ec2\_example" {

ami = "ami-0fc5d935ebf8bc3bc"

instance\_type = "t2.micro"

key\_name = "tfkey"

connection{

type = "ssh"

host = self.public\_ip

user = "ubuntu"

private\_key = file("/home/aishwaryawaghma/key/tfkey")

timeout = "4m"

}

} [7]

resource "aws\_security\_group" "main" {

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

resource "aws\_key\_pair" "deployer" {

key\_name = "tfkey"

public\_key = " ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDOkZjyYKt19Om7cl6xkwyB5bIDt3Qj+AQFyp1ilee1Akk3Gca2pZRxGrMKh2iQMlwNf5xwmMqUY+rXRxEM0iC68SXM76JfrHXtj/ImPpdxymEGV5i4I+Ch6BlqxyHs2lC8DlgBp8byamyJq0Td3TnRlYFwLZWwZZHC5iQEVFcOtZUk2b8e9xGw0+p5iJP5DkvrlQEY4bsypw5JEspe1Ead+0aiUIdB

tB4SBwb5odwUwqYh508SoDScQxzjGeKub6G912ynMBQ+hgS0xLvhg0cWF0nKo9wqu7yGTa+20x2TicKVODa2S6FdsPxyT8cZwpLkns0r2RRjUGnyoitRDG2n root@ip-172-31-42-5”}

**$ sudo terraform init**

Initializing the backend...

Initializing provider plugins...

- Finding latest version of hashicorp/aws...

- Installing hashicorp/aws v5.30.0...

- Installed hashicorp/aws v5.30.0 (signed by HashiCorp)

Terraform has been successfully initialized as shown in figure 4.

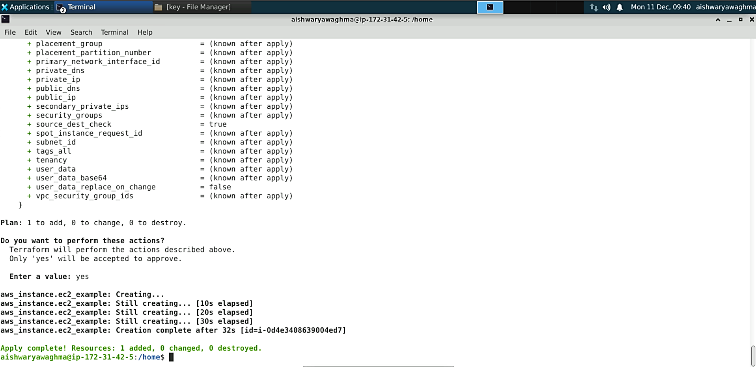


Figure 4: Terraform initialization in Ubuntu

**$ sudo terraform plan**

Plan: 3 to add, 0 to change, 0 to destroy.

**$ sudo terraform apply**

aws\_key\_pair.deployer: Refreshing state... [id=tfkey]

aws\_security\_group.main: Refreshing state... [id=sg-0c8786946377764af]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols [7]:

+ create

Terraform will perform the following actions:

# aws\_instance.ec2\_example will be created

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value: yes

aws\_instance.ec2\_example: Creating...

aws\_instance.ec2\_example: Still creating... [10s elapsed]

aws\_instance.ec2\_example: Still creating... [20s elapsed]

aws\_instance.ec2\_example: Still creating... [30s elapsed]

aws\_instance.ec2\_example: Creation complete after 32s [id=i-0d4e3408639004ed7]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

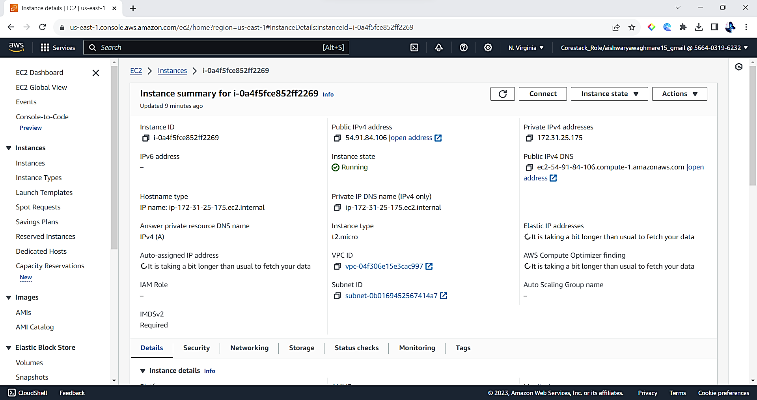


Figure 5: EC2 Instance created in AWS Console

**$ cd /home/aishwaryawaghma/key**

**$ sudo su**

**root@ip-172-31-42-5:/home/aishwaryawaghma/key# chmod 400 tfkey.pem**

**root@ip-172-31-42-5:/home/aishwaryawaghma/key# ssh -i "tfkey.pem" ubuntu@ec2-54-91-84-106.compute-1.amazonaws.com**

The authenticity of host 'ec2-54-91-84-106.compute-1.amazonaws.com (54.91.84.106)' can't be established.

ECDSA key fingerprint is SHA256:H8snJt1RkSwySygjxBl9EygU9Ps89NbEyb31BkybIZo.

Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

Warning: Permanently added 'ec2-54-91-84-106.compute-1.amazonaws.com,54.91.84.106' (ECDSA) to the list of known hosts.

Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-1012-aws x86\_64) [7].

**ubuntu@ip-172-31-25-175:~$ sudo apt-get update**

**ubuntu@ip-172-31-25-175:~$ sudo apt install default-jre**

**ubuntu@ip-172-31-25-175:~$ sudo apt-get install ansible**

**ubuntu@ip-172-31-25-175:~$ wget -qO - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -**

OK

**ubuntu@ip-172-31-25-175:~$ sudo apt-get update**

**ubuntu@ip-172-31-25-175:~$ sudo apt-get install jenkins**

**ubuntu@ip-172-31-25-175:~$ sudo systemctl status jenkins**

● jenkins.service - Jenkins Continuous Integration Server

Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)

**Active: active (running)** since Mon 2023-12-11 10:28:07 UTC; 1min 1s ago [8].

Main PID: 9074 (java)

Tasks: 43 (limit: 1121)

Memory: 326.7M

CPU: 43.723s

CGroup: /system.slice/jenkins.service

└─9074 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

ubuntu@ip-172-31-25-175:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword

93a039b76b8c4f1aa2280f6880708b71 [7].

**Jenkins URL –** <http://54.91.84.106:8080/>

A screenshot of a computer

Description automatically generated

Figure 6: Adding port 8080 for Jenkins

**ubuntu@ip-172-31-25-175:~$ sudo apt update**

**ubuntu@ip-172-31-25-175:~$ sudo apt install software-properties-common**

**ubuntu@ip-172-31-25-175:~$ sudo add-apt-repository ppa:deadsnakes/ppa**

**ubuntu@ip-172-31-25-175:~$ sudo apt update**

**ubuntu@ip-172-31-25-175:~$ sudo apt install -y python3-pip**

**ubuntu@ip-172-31-25-175:~$ sudo apt-get install python-is-python3**

**ubuntu@ip-172-31-25-175:~$ python --version**

Python 3.10.12 [8]

A screenshot of a computer

Description automatically generated

Figure 7: Jenkins using Terraform Automation

* 1. **Case 2: Building a Docker Jenkins Pipeline for CI/CD Workflow**

The second experiment aimed to showcase the efficacy of continuous integration (CI) and continuous delivery (CD) workflows through the creation of a Docker Jenkins Pipeline [10].

**Tools/ Methodology:**

1. Docker: Utilized for constructing the application from a Dockerfile and subsequently pushing this build to Docker Hub.
2. Docker Hub: Served as the repository for storing the Dockerized application images.
3. GitHub: Acted as the version control system for maintaining the application's codebase and tracking changes as mentioned in figure 8, 9, 10.
4. Git: Enabled the synchronization of local code updates with the GitHub repository.
5. Linux (Ubuntu): Provided the foundational operating system environment necessary for initiating and carrying out the project tasks.
6. Jenkins: Facilitated the automation of the deployment processes inherent to the continuous integration efforts [11].

**Procedure:** The procedure involved meticulous documentation of the installation and configuration stages, from inception to completion. Key steps included maintaining an up-to-date codebase within GitHub, ensuring the application's and its versions' availability on Docker Hub, and monitoring Jenkins' build status for each incremental update. This process culminated in the development of a Docker Jenkins Pipeline, illustrating a seamless CI and CD flow. The pipeline's responsibilities encompassed generating a Docker image from the Dockerfile, hosting this image on Docker Hub, and enabling the image's deployment as a Docker container.

**Execution:** Careful assembly of the Docker Jenkins Pipeline was paramount to validating the continuous integration and delivery mechanisms' functionality [9][10]

**Source Code:**

**ubuntu@ip-172-31-25-175:~$ java -version**

openjdk version "11.0.21" 2023-10-17

OpenJDK Runtime Environment (build 11.0.21+9-post-Ubuntu-0ubuntu122.04)

OpenJDK 64-Bit Server VM (build 11.0.21+9-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)

**ubuntu@ip-172-31-25-175:~$ sudo systemctl status** Jenkins

A screenshot of a computer

Description automatically generated

Figure 8: Jenkins Installation

A screenshot of a computer

Description automatically generated

Figure 9: Adding new SSH key in GitHub

A screenshot of a computer

Description automatically generated

Figure 10: Creating a repository in GitHub

jenkins.service - Jenkins Continuous Integration Server

Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)

Active: active (running) since Mon 2023-12-11 10:50:07 UTC; 1h 57min ago

Main PID: 12345 (java)

Tasks: 46 (limit: 1121)

Memory: 537.5M

CPU: 1min 57.207s

CGroup: /system.slice/jenkins.service

└─12345 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Webhooks are a powerful tool for enabling real-time communication and automation between web applications and services, enhancing interoperability and enabling seamless integration between disparate systems. Webhooks are a mechanism for automatically notifying or triggering actions in a web application when certain events occur. Instead of a user actively polling or checking for updates, the application delivers real-time data to a specified URL endpoint whenever a specific event occurs. This enables seamless integration and automation between different systems or services. Event Occurs: A specific event, such as a new user signup, a payment received, or a file uploaded, occurs within an application or service. The application generates a webhook event payload containing relevant data about the event and sends it as an HTTP POST request to a preconfigured URL endpoint. The receiving endpoint, typically owned by another application or service, receives the webhook payload. The receiving application processes the webhook payload, performs any necessary actions or logic based on the event data, and may respond with a status code indicating successful processing. Based on the processing, the receiving application may take actions such as updating a database, sending notifications, triggering workflows, or initiating further processing. The managing the webhooks are represented in figure 11. Webhooks are commonly used for various purposes, including:

* Integration: Connecting different applications or services to automate workflows and data synchronization.
* Real-time Notifications: Notifying external systems or users immediately when specific events occur.
* Automation: Triggering actions or processes automatically in response to events, such as updating records, sending notifications, or executing scripts.
* Data Syncing: Keeping data synchronized between different systems or databases in real-time.
* Customization: Allowing developers to extend or customize the functionality of applications by reacting to specific events.

A screenshot of a computer

Description automatically generated

Figure 11: Managing Webhooks in GitHub

sudo apt-get install \ ca-certificates \ curl \ gnupg \ lsb-release

**ubuntu@ip-172-31-25-175:~$ curl -fsSL** https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archivekeyring.gpg

A screenshot of a computer

Description automatically generated

Figure 12: Adding repository URL in Jenkins

**ubuntu@ip-172-31-25-175:~$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archivekeyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**

A screenshot of a computer

Description automatically generated

Figure 13: Building Triggers in Jenkins

In Jenkins, triggers are mechanisms that initiate a build process in response to specific events or conditions. The process of building triggers is represented in figure 13. There are various types of triggers available in Jenkins, each serving different purposes. Here are some common triggers and how to set them up:

1. SCM Polling Trigger: This trigger polls the Source Code Management (SCM) system at regular intervals to check for changes. If changes are detected, a build is triggered. To set up SCM polling:

* Navigate to your Jenkins job configuration.
* Under the "Build Triggers" section, select "Poll SCM."
* Specify the polling schedule using cron syntax (e.g., H/5 \* \* \* \* for polling every 5 minutes).
* Save your configuration.
* GitHub/Bitbucket/GitLab Hook Trigger: This trigger listens for webhook notifications from your version control system (e.g., GitHub, Bitbucket, GitLab) and triggers a build when code changes are pushed.

1. To set up webhook triggers:

* Install the appropriate plugin for your version control system (e.g., GitHub Plugin, Bitbucket Branch Source Plugin).
* Configure the webhook URL in your version control system to point to your Jenkins server.
* Enable the "GitHub hook trigger for GITScm polling" or similar option in your Jenkins job configuration under "Build Triggers."
* Build Periodically Trigger: This trigger initiates builds at specified time intervals.

1. To set up periodic builds:

* Navigate to your Jenkins job configuration.
* Under the "Build Triggers" section, select "Build periodically."
* Specify the schedule using cron syntax (e.g., H/15 \* \* \* \* for building every 15 minutes).
* Save your configuration.
* Manual Trigger: This trigger allows builds to be started manually by users.

1. To set up manual builds:

* Navigate to your Jenkins job configuration.
* Under the "Build Triggers" section, deselect all automated triggers.
* Enable the "Build after other projects are built" option if you want manual builds to be triggered after specific upstream jobs.
* Save your configuration.
* Parameterized Trigger: This trigger allows builds to be started with parameters passed to them.’

To set up parameterized builds:

* Navigate to your Jenkins job configuration.
* Under the "Build Triggers" section, select "Build after other projects are built."
* Optionally, specify parameters to be passed to the downstream job.
* Save your configuration.
* These are just a few examples of triggers available in Jenkins. Depending on your requirements, you can mix and match these triggers or use them in combination to achieve the desired build automation and orchestration.

The following procedure mentions the docker build, creating docker repository also as represented in figure 14,15 and 16.

**ubuntu@ip-172-31-25-175:~$ sudo apt-get update**

**ubuntu@ip-172-31-25-175:~$ sudo apt-get install docker-ce-cli**

**ubuntu@ip-172-31-25-175:~$ docker --version**

Docker version 24.0.7, build afdd53b

**ubuntu@ip-172-31-25-175:~$ git --version**

git version 2.34.1

[11] **ubuntu@ip-172-31-25-175:~$ ssh-keygen -t rsa -b 4096 -C "aishwaryawaghmare15@gmail.com"**

Generating public/private rsa key pair.

Enter file in which to save the key (/home/ubuntu/.ssh/id\_rsa):

ubuntu@ip-172-31-25-175:~$ cat /home/ubuntu/.ssh/id\_rsa.pub

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCv+omXdg3SmfFmnLetHqQw9sVZj0bw0Tnvf7Rlm/DshY7KNBlIltfyYW98rJRbr2MdeQOx/wCjQ53ucMXF0NKKcqA2XrSDZKPcFqtWGEiLuz+whX4gZ610DBB0bdJGq/SwynY8WekVfX41yIzUQRFBtqDzsEZuZkLWq4juQxoDG8zDFpfOdQT1/IGo06N0Ixp4La0TLqwHqtsntJRHlcm0ZVn2BsWQWYRaY8E2

lAPEhTgFUTuGmRBmXpwJgNe+PP/fnKlBJ7ts/XBpIMobfNyVpYjsm0Ls5b4XSTxb2cvqwGtuHiCD600K1wh40LpQ1jPWkTBMb0h0mdUoksHaLp75bTjdC5zejyDqxedfUZOfun9HmzoNHGLoZPMCmeja1rdNG/KHmjnNlxyVz/F8hpg0v0WQtHQ6KDt/dXBA8HHSjb0xrM5gla/5GATY4bskf4l6wRP9oXyqb5tw6jWDcBeuNdoGRw0uPR8s7NexSeaO2ullbn/5LN4R3xDFfw

yGkLl5GlxrzF1GbByeR6+d5VYq1y+IZJcw5ISUBR2JYUm7e/GCgXb8BN/ZrI9xFSsne5Mjh1tBNqnkWfNPttrK/FUfTegdSmgCOiHMaCiH1w2JUhLT096aJftzjZRz9AL4v6W4CMZu+qjBk8cv6Ye4Uz58MW8wQ7yoWEP0r49M6Om0jw== aishwaryawaghmare15@gmail.com

**ubuntu@ip-172-31-25-175:~$ mkdir jenkinsdocker**

**ubuntu@ip-172-31-25-175:~$ cd jenkinsdocker/**

**ubuntu@ip-172-31-25-175:~/jenkinsdocker$ ls**

**ubuntu@ip-172-31-25-175:~/jenkinsdocker$ sudo docker build --tag jenkinsdocker .**

[+] Building 0.2s (2/2) FINISHED

A screenshot of a computer

Description automatically generated

Figure 14: Docker Build and Publish in Jenkins

docker:default

=> [internal] load build definition from Dockerfile 0.1s

=> => transferring dockerfile: 2B 0.0s

=> [internal] load .dockerignore 0.1s

=> => transferring context: 2B

A screenshot of a computer

Description automatically generated

Figure 15: Creating repository in Docker Hub

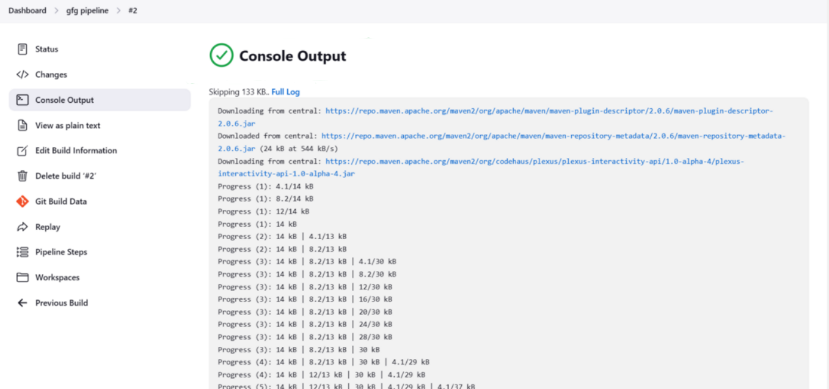


Figure 16: Implementation of CI/ CD in Jenkins

**c. Case 3: Proposed solution**

The objective of this work is to develop an advanced chatbot that proactively alerts engineers about significant fluctuations in Key Performance Indicators (KPIs) and facilitates the creation of a detailed KPI tracking and analytics dashboard. This innovative system employs state-of-the-art Natural Language Processing (NLP) and deep learning technologies to precisely interpret customer intentions and efficiently filter logs, ensuring a seamless user experience.

1. **Configurations**
2. The Python is selected for its extensive support for AI and machine learning libraries, making it the ideal choice for chatbot development. Other languages such as Java, C, C#, and C++ can be considered based on specific project requirements and team expertise.
3. Bot Framework Selection: Rasa: Preferred for its flexibility and scalability, Rasa offers a comprehensive framework for building sophisticated chatbots. Microsoft Bot Framework: Considered for its seamless integration capabilities with other Microsoft services, providing a robust platform for chatbot development.
4. NLP Integration: Core Libraries: Utilize spaCy for efficient language processing and Hugging Face Transformers for access to advanced language models.
5. Key Capabilities: Implement tokenization to break down user queries into manageable pieces, named entity recognition to identify important elements in the text, and sentiment analysis to gauge the emotional tone of user queries.
6. Deep Learning Model Integration: Frameworks: TensorFlow or PyTorch are used for integrating deep learning models, providing a solid foundation for advanced AI capabilities. Architectures: Explore recurrent neural networks (RNNs) for their ability to process sequential data and transformers for handling complex language patterns with attention mechanisms. Intent Recognition: Enhance the chatbot's ability to recognize user intentions with precision, leveraging deep learning models for accurate interpretation of user queries.
7. **Frontend Interface Development: UI Framework Selection**
   1. React.js: Selected for its ability to build dynamic and responsive user interfaces, ensuring a smooth and interactive experience for users. Component Library: Material-UI or Bootstrap is used for ready-to-use UI components, providing a consistent and attractive design language. Styling: CSS Frameworks: Tailwind CSS is employed for its utility-first approach, enabling rapid development of custom-designed interfaces. Responsive Design: Ensure the interface adapts seamlessly to different devices and screen sizes, providing an optimal viewing experience across all platforms.
   2. Data Visualization: Chart Libraries: Chart.js or High charts are used for creating visually appealing and interactive charts, enabling users to easily understand and analyze KPI data.
   3. Customization: Customize the appearance of charts, including color, labels, and tooltips, to enhance readability and provide a better user experience. Data Retrieval: API Communication: Axios is utilized for making HTTP requests to backend APIs, ensuring efficient retrieval of KPI data. State Management: React hooks or Redux are employed for managing the state of the data, ensuring that the frontend reflects real-time updates and changes.
8. **KPI Monitoring, Database, Analytics Dashboard, and Backend Frameworks**
9. Backend Server Implementation: Frameworks: Express.js is chosen for its lightweight nature and flexibility, while Django is selected for its high-level features and robustness. API Design: Design RESTful APIs that provide a clear and consistent interface for communication between the frontend and backend, ensuring smooth data exchange.
10. Analytics Dashboard Development: Visualization Tools: Plotly Dash is used for creating Python-based interactive dashboards, while Tableau is employed for its powerful data visualization capabilities. Interactivity: Incorporate interactive elements such as filters, dropdowns, and date pickers in the dashboard, allowing users to customize their view of KPI data and gain deeper insights.
11. Database Management: Database Selection: Choose between PostgreSQL for a reliable and scalable relational database or MongoDB for a flexible NoSQL database, depending on the data structure and scalability requirements.
12. Data Modelling: Develop a data model that efficiently organizes KPI data and other relevant information, ensuring quick retrieval and analysis. System Monitoring: Monitoring Tools: Grafana is used for customizable dashboards that provide real-time insights into system performance, while Prometheus is employed for its powerful monitoring and alerting capabilities.
13. Alert Configuration: Set up alerts for KPI anomalies to notify engineers immediately of any significant changes, enabling proactive response to potential issues.
14. **Deployment, Version Control, Continuous Integration, and Delivery**
15. Deployment Options: Cloud Services: AWS, Azure, or Google Cloud are considered for their scalability, reliability, and wide range of services.
16. On-Premises: On-premises servers are evaluated for scenarios where greater control over the infrastructure is required.
17. Containerization: Docker: Utilized for creating lightweight and portable containers for the application components, ensuring consistency across different environments.
18. Kubernetes: Employed for orchestrating and managing containers, providing automated scaling, deployment, and management of containerized applications.
19. CI/CD Pipelines: Automation Tools: Jenkins or GitLab CI are used for automating the testing and deployment processes, ensuring that the codebase is continuously integrated and delivered with high quality.
20. Pipeline Configuration: Configure pipelines to automatically build, test, and deploy code changes, streamlining the development workflow and reducing manual intervention.
21. Version Control: Git: Adopted for its powerful version control capabilities, allowing developers to track changes, collaborate on code, and manage different versions of the project. Branching Strategy: Implement a branching strategy such as Git Flow to manage feature development, releases, and hotfixes in an organized manner.
22. **Documentation and Additional Considerations**
23. Comprehensive Documentation:Code Documentation: Employ tools like JSDoc or Sphinx to document the codebase and API endpoints thoroughly, providing clear guidance for developers and users.System Architecture: Create detailed diagrams and documentation to describe the overall architecture of the system, including data flow and component interactions, ensuring a clear understanding of the system's design.Security Measures:
24. Encryption: Implement encryption techniques to secure sensitive data both at rest and in transit, protecting against unauthorized access and data breaches. Authentication: Utilize JSON Web Tokens (JWT) for securing API endpoints, ensuring that only authenticated users can access the system's functionalities.
25. Scalability and Performance: Load Balancing: Implement load balancers to distribute incoming traffic evenly across servers, preventing any single server from becoming overwhelmed and ensuring high availability. Caching: Employ caching mechanisms to store frequently accessed data temporarily, reducing database load and improving response times for a better user experience.
26. **Application Areas**

The chatbot can be tailored to meet the specific needs of various sectors, including

1. Financial Sector: Employ the chatbot for detecting and alerting on fraudulent activities such as phishing and identity theft, enhancing security for financial institutions. Healthcare Sector: Utilize the chatbot to protect sensitive patient information and aid in the prevention of data breaches, ensuring compliance with regulations and safeguarding patient privacy.
2. E-Commerce Sector: Implement the chatbot to prevent online fraud and secure customer data, providing a safe and trustworthy shopping experience for users.
3. Education Sector: Deploy the chatbot to safeguard student and faculty data, enhancing cybersecurity measures and preventing unauthorized access to educational records. Government Sector: Leverage the chatbot to secure sensitive government data, improve response to cyber threats, and ensure the integrity of government operations.
4. **RESULTS AND ANALYSIS**

An analytics dashboard is a visual representation of data and key performance indicators (KPIs) that provides insights into various aspects of an organization's operations, performance, and trends. It consolidates data from multiple sources and presents it in an easily understandable format, allowing stakeholders to make informed decisions based on data-driven insights. Here are some key components presented using analytics dashboards represented in figures 17 to 22.

We have utilized charts, graphs, and other visual elements to present data in a visually appealing and easy-to-understand manner. Common types of visualizations include bar charts, line graphs, pie charts, heatmaps, and maps.

It is possible to customize dashboards which allows to customize the layout, content, and appearance of the dashboard according to their preferences and specific requirements. This may include the ability to rearrange widgets, change colors, and adjust data filters.

Other interactive elements incorporate such as filters, drill-down capabilities, tooltips, and hover effects to enable users to explore and analyze data in more detail.

The real-time data updates provide real-time or near-real-time updates to ensure that users have access to the latest information and insights. This may involve automatic data refreshes at regular intervals or triggered updates based on specific events. The Key Performance Indicators (KPIs) highlights important metrics and KPIs relevant to the organization's goals and objectives. KPIs may include metrics related to sales, marketing, finance, operations, customer service, and more.

The alerts and notifications allow users to set up alerts and notifications based on predefined thresholds or conditions. This enables proactive monitoring and timely responses to important changes or anomalies in the data. The role-based Access Control implements role-based access control (RBAC) to restrict access to sensitive or confidential data and features based on users' roles and permissions within the organization.

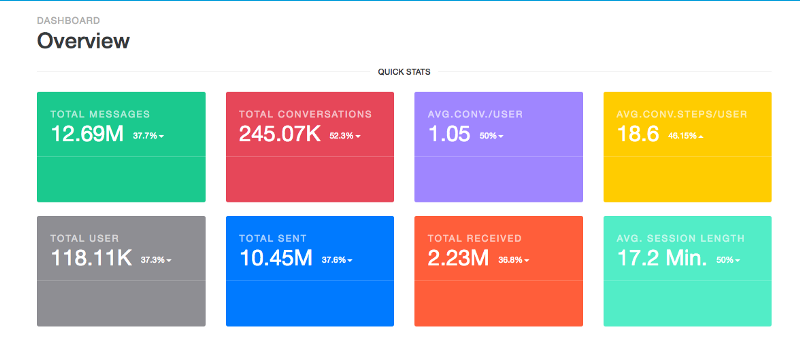


Figure 17: Dashboard

**A screenshot of a computer

Description automatically generated**

Figure 18: Analytics Dashboard

**A screenshot of a computer

Description automatically generated**

Figure 19: Summary and AI Powered Insights

**A screenshot of a computer

Description automatically generated**

Figure 20: Dashboard Output Console

**A screenshot of a computer

Description automatically generated**

Figure 21: Results and Findings

**A screen shot of a graph

Description automatically generated**

Figure 22: Charts and Figures

1. **CONCLUSION**

This research offers a thorough examination of the development process for a chatbot aimed at maintaining circuit uptime at 99.9% and minimizing the necessity for direct engineer-client communication during early troubleshooting phases. The study weaves together literature reviews, case studies, and potential solutions, showcasing the transformative potential of chatbots, particularly those enhanced with deep learning, across various societal sectors. The development journey of the chatbot entailed venturing into web development and programming languages to forge a platform that integrates cybersecurity measures, authentication, authorization, alongside artificial intelligence, and machine learning, all within the frameworks of DevSecOps and DevOps. While the chatbot technology remains in developmental stages, its future implications appear vast as the technology matures. As chatbots evolve, their application spectrum broadens, extending from basic customer service roles to complex tasks across banking, agriculture, and healthcare sectors, to name a few. The significant potential impact of chatbots in these fields is undeniable, with ongoing research likely to usher in more advanced, capable chatbots. This study underscores chatbots' capacity to revolutionize various societal segments, offering more efficient and effective task management solutions.

Future enhancements and directions for the chatbot development includeAdvanced NLP Techniques incorporate cutting-edge NLP techniques such as context-aware language understanding and advanced sentiment analysis to improve the chatbot's comprehension and interaction capabilities.The other directions considered as use of deep learning model for refinement which continuously refine and update deep learning models with new datasets and architectures to enhance the chatbot's accuracy and adaptability to evolving user needs. The other direction such as cybersecurity enhancements including exploring advanced cybersecurity measures like blockchain technology for secure transactions and biometric authentication for enhanced access control, ensuring a higher level of security for the system.

Also it is necessary to focus on optimizing the performance of the chatbot through efficient resource management, cloud computing services, and continuous testing and deployment practices, ensuring the chatbot remains responsive and reliable under varying workloads.

REFERENCES

1. A. Alshamrani and A. Bahattab, “A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model,” IJCSI International Journal of Computer Science, vol. 12, no. 1, p. 6, 2015.
2. S. Balaji and D. Murugaiyan, “WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC,” International Journal of Information Technology and Business Management, vol. 2, p. 5, 2012.
3. Simplilearn, “What is Agile: Understanding Agile Methodology and Principles in Software Development?,” Simplilearn, 20 03 2024. [Online]. Available: https://www.simplilearn.com/tutorials/agile-scrum-tutorial/what-is-agile. [Accessed 04 04 2024].
4. P. Raj and D. P. Sinha, “Project Management In Era Of Agile And Devops Methodlogies,” INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH , vol. 9, p. 10, 2020.
5. M. A. Marwan and A. Achdian, “Analysis of CI/CD Application Based on Cloud Computing Services on Fintech Company,” International Research Journal of Advanced Engineering and Science, vol. 4, no. 3, p. 3, 2019.
6. A. Alsanad, S. Mahmood, K. Smolander and M. A. Akbar, “Toward successful DevSecOps in software development organizations: A decision-making framework,” Information and Software Technology, vol. 147, no. 106894, p. 21, 2022.
7. J. Nawagamuwa, INFRASTRUCTURE AS CODE FRAMEWORKS EVALUATION FOR SERVERLESS APPLICATIONS TESTING IN AWS, Tampere: Master of Science Thesis, 2023.
8. M. Howard, Terraform — Automating Infrastructure As A Service, Portland: Michael Howard P.Eng, 2022.
9. R. Leszko, Continuous Delivery with Docker and Jenkins: Create secure applications by building complete CI/CD pipelines, Birmingham: Packt Publishing Ltd, 2022.
10. D. Edwards and B. Atkinson, Generic Pipelines Using Docker: The DevOps Guide to Building Reusable, Platform Agnostic CI/CD Frameworks, Apress, 2018, 2018.
11. K. Subrahmanyam, D. R. Rao and M. K. Abhishek, “Framework to Deploy Containers using Kubernetes and CI/CD Pipeline,” (IJACSA) International Journal of Advanced Computer Science and Applications, vol. 13, no. 4, p. 5, 2022.
12. J. Lee, “The practical guide to chatbot metrics and analytics,” The Startup, 29 June 2018. [Online]. Available: https://medium.com/swlh/the-practical-guide-to-chatbot-metrics-and-analytics-2a734064e833. [Accessed 04 April 2024].