**VARUVAN VADIVELAN INSTITUTE OF TECHNOLOGY**

**NAAN MUDHALAVAN CLOUD APPLICATION DEVELOPMENT PHASE 3**

**PROJECT TITLE: CHATBOT DEPLOYMENT WITH IBM CLOUD WATSON ASSISTANT**

**DEVELOPMENT PART**

Sure, let's dive into the development part of chatbot deployment. This process typically involves several key steps:

1. \*\*Define Objectives and Use Cases:\*\*

- Clearly outline the purpose of your chatbot and the specific tasks it should perform.

- Identify the target audience and understand their needs.

2. \*\*Choose a Platform:\*\*

- Decide where your chatbot will live—on a website, messaging app, or a custom app.

- Select a platform or framework that aligns with your development skills and deployment goals. Popular choices include Dialogflow, Microsoft Bot Framework, or custom development using frameworks like Django or Flask for Python.

3. \*\*Design Conversational Flow:\*\*

- Create a flowchart or diagram that outlines how the conversation will progress.

- Define possible user inputs and map out the corresponding bot responses.

4. \*\*Natural Language Processing (NLP):\*\*

- If your chatbot involves understanding and generating human-like language, implement Natural Language Processing.

- Train your model to recognize intents and entities within user input.

5. \*\*Integration with Backend Services:\*\*

- Connect your chatbot to relevant backend services or databases to fetch information or perform actions.

- Ensure secure communication between the chatbot and other systems.

6. \*\*Development and Testing:\*\*

- Write the code for your chatbot based on the design and specifications.

- Test the chatbot extensively to identify and fix any bugs.

- Incorporate user feedback to improve the user experience.

7. \*\*Deploy the Chatbot:\*\*

- Choose a hosting environment for your chatbot. This could be cloud platforms like AWS, Azure, or Google Cloud.

- Deploy your chatbot application to make it accessible to users.

8. \*\*Monitoring and Analytics:\*\*

- Implement monitoring tools to track the performance of your chatbot.

- Use analytics to gather insights into user interactions and continuously improve the chatbot's capabilities.

9. \*\*Security Measures:\*\*

- Implement security protocols to protect user data and ensure secure communication.

- Regularly update and patch your chatbot to address potential vulnerabilities.

10. \*\*User Training and Documentation:\*\*

- Provide clear instructions for users on how to interact with the chatbot.

- Consider implementing user training within the chatbot to guide users through the conversation.

Remember, the development process is iterative, and it's essential to gather feedback from real users to make continuous improvements. How do you feel about this process?

CHATBOT CLASSIFICATION:

Classifying chatbots can be done based on various criteria. Here are a few common ways to classify them:

\*\*Rule-Based vs. AI-Powered:\*\*

- \*\*Rule-Based Chatbots:\*\* Follow a set of predefined rules. They are good for specific and structured tasks.

- \*\*AI-Powered Chatbots:\*\* Use machine learning and natural language processing to understand and respond to user inputs. They can handle more complex and dynamic conversations.

**PROGRAM**

from flask import Flask, request, jsonify

app = Flask(\_\_name\_\_)

def respond\_to\_user(input\_text):

    # Simple rule-based logic to determine chatbot responses

    if "hello" in input\_text.lower():

        return "Hello! How can I help you today?"

    elif "bye" in input\_text.lower():

        return "Goodbye! Have a great day."

    else:

        return "I'm not sure how to respond to that. Can you please provide more details?"

@app.route('/chat', methods=['POST'])

def chat():

    user\_input = request.json.get('user\_input', '')

    chatbot\_response = respond\_to\_user(user\_input)

    return jsonify({'response': chatbot\_response})

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(port=5000)

**OUTPUT**

**I'm just text on a screen, so I can't execute programs. However, I can guide you on how you might run and test the provided Python program.**

**1. Save the code in a file, for example, `chatbot.py`.**

**2. Open a terminal or command prompt.**

**3. Navigate to the directory where you saved `chatbot.py`.**

**4. Run the following command to start the Flask serve**

I'm just text on a screen, so I can't execute programs. However, I can guide you on how you might run and test the provided Python program.

1. Save the code in a file, for example, `chatbot.py`.

2. Open a terminal or command prompt.

3. Navigate to the directory where you saved `chatbot.py`.

4. Run the following command to start the Flask server:

   ```bash

   python chatbot.py

   ```

   If you're using Python 3, you might need to use `python3` instead.

5. Once the server is running, you can use a tool like `curl` or Postman to send a POST request to the `/chat` endpoint. For example:

   ```bash

   curl -X POST -H "Content-Type: application/json" -d '{"user\_input": "Hello, chatbot!"}' http://localhost:5000/chat

   ```

   This should return a JSON response with the chatbot's reply

If you encounter any issues or have questions about specific parts of the

code, feel free to ask!A

**CLIENT USER INTERFACE**

Creating a chatbot client user interface involves designing a front-end application that allows users to interact with the chatbot. Let's consider a simple example using HTML, CSS, and JavaScript to create a basic chat interface.

```html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Simple Chatbot</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            margin: 0;

            padding: 0;

            background-color: #f4f4f4;

        }

        #chat-container {

            max-width: 400px;

            margin: auto;

            padding: 20px;

            background-color: #fff;

            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

            border-radius: 8px;

            margin-top: 50px;

        }

        #messages {

            list-style-type: none;

            padding: 0;

            margin: 0;

            overflow-y: auto;

            max-height: 300px;

            border-bottom: 1px solid #ccc;

        }

        #user-input {

            width: 100%;

            padding: 10px;

            box-sizing: border-box;

        }

    </style>

</head>

<body>

    <div id="chat-container">

        <ul id="messages"></ul>

        <input type="text" id="user-input" placeholder="Type your message...">

        <button onclick="sendMessage()">Send</button>

    </div>

    <script>

        function appendMessage(message, isUser = false) {

            const messagesContainer = document.getElementById('messages');

            const li = document.createElement('li');

            li.textContent = message;

            li.style.textAlign = isUser ? 'right' : 'left';

            messagesContainer.appendChild(li);

        }

        function sendMessage() {

            const userInput = document.getElementById('user-input').value;

            appendMessage('You: ' + userInput, true);

            // You can replace the following code with actual API calls to your chatbot server

            fetch('http://localhost:5000/chat', {

                method: 'POST',

                headers: {

                    'Content-Type': 'application/json',

                },

                body: JSON.stringify({

                    user\_input: userInput,

                }),

            })

            .then(response => response.json())

            .then(data => {

                const chatbotResponse = data.response;

                appendMessage('Chatbot: ' + chatbotResponse);

            })

            .catch(error => console.error('Error:', error));

        }

    </script>

</body>

</html>

```

**SERVICE INSTANCE**

It seems like you're asking about a program that involves interacting with a service instance. The specifics would depend on the type of service you're referring to—this could be a web service, a database, an API, or any other service that your program interacts with.

**PROGRAM**

import requests

def fetch\_data\_from\_service(api\_endpoint):

    try:

        response = requests.get(api\_endpoint)

        if response.status\_code == 200:

            data = response.json()

            return data

        else:

            print(f"Error: {response.status\_code}")

    except Exception as e:

        print(f"An error occurred: {e}")

# Example usage

api\_endpoint = "https://api.example.com/data"

result = fetch\_data\_from\_service(api\_endpoint)

if result:

    print("Data received:")

    print(result)

else:

    print("Failed to fetch data.")

**OUTPUT**

pip install requests

**KUBERNETES USING CLOUD SERVICE**

Creating a Kubernetes application involves several steps, and it depends on what you want the application to do. Below is a simple example of a Kubernetes Deployment and Service using a Python Flask application. This example assumes you have a Kubernetes cluster set up and the **kubectl** command-line tool installed.

# app.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

    return "Hello, Kubernetes!"

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, host='0.0.0.0')

CREATE DOCKERFILE

# Dockerfile

FROM python:3.8

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

COPY . .

CMD ["python", "app.py"]

**Create a Kubernetes Deployment (deployment.yaml):**

# deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: flask-app

spec:

  replicas: 3

  selector:

    matchLabels:

      app: flask-app

  template:

    metadata:

      labels:

        app: flask-app

    spec:

      containers:

      - name: flask-app

        image: your-docker-username/your-flask-app:latest

        ports:

        - containerPort: 5000

SERVICE.yaml;

# service.yaml

apiVersion: v1

kind: Service

metadata:

  name: flask-app-service

spec:

  selector:

    app: flask-app

  ports:

    - protocol: TCP

      port: 80

      targetPort: 5000

  type: LoadBalancer