**Problem: For creating issues one cannot always login into the jira Account and rise the issue, hence this process needs to be automated and here we are integrating it with GitHub where if we commented /jira in GitHub a jira ticket needs to be created and in order to communicate between GitHub and Jira webhooks are utilised and for the logic part python programming is used. Configure a webhook, where URL is provided by Devops Engineer required for the python application.**

**Tasks:**

**Task1:** Jira setup -- choose Scrum project – to interact with applications, API is used and API token is utilized

**Task2:** Jira API calls

**Task3:** Python script

**Task4:** Execution

**Jira Ticketing and Integration with GitHub**

**1. Overview of Jira Ticketing**

Jira, developed by Atlassian, is a widely used project management and issue-tracking platform, particularly popular in agile software development environments. A **Jira ticket** represents a single unit of work, which could be a user story, bug, task, or epic. These tickets facilitate structured project tracking and effective collaboration among development, testing, and management teams.

Each Jira ticket typically includes:

* **Issue Key and Summary**: A unique identifier and concise title.
* **Description**: Detailed information about the task or issue.
* **Status**: Workflow states such as To Do, In Progress, and Done.
* **Assignee and Reporter**: Responsible and reporting team members.
* **Priority**: Level of urgency.
* **Sub-tasks and Linked Issues**: Dependencies and related tasks.
* **Activity Log**: Comments, work logs, and change history.

This system provides transparency, accountability, and traceability throughout the development lifecycle.

**2. Integration of Jira with GitHub**

The integration of Jira with GitHub streamlines the software development process by linking code activity to project tasks. This ensures developers can track their progress directly from commits and pull requests, while project managers gain visibility into the development status without switching platforms.

**2.1 Purpose and Benefits**

* **Improved Traceability**: GitHub commits, branches, and pull requests (PRs) can be directly associated with Jira tickets.
* **Automation**: Status updates in Jira can be triggered by GitHub events (e.g., PR merged).
* **Enhanced Collaboration**: Both developers and stakeholders have visibility into ongoing work.
* **Centralized Activity**: Jira tickets display relevant development data such as commit messages, branch names, and PR links.

**What is a Webhook?**

A **webhook** is a way for one app or service to send **real-time data** to another app when something happens.

Think of it like this:

“If something happens in App A, send the details to App B automatically.”

It's like **setting up a doorbell** — when someone presses it (an event happens), the bell rings (another action is triggered).

**How It Works (Step-by-Step)**

1. **You give App A a URL** (like your website address).
2. When something happens in App A (e.g., a user creates a pull request),
3. App A sends a **message (HTTP POST request)** to the URL you provided.
4. That message includes **details** about what happened.
5. Your app receives that message and does something with it (e.g., updates a database, sends an email, etc.).

**Real-Life Examples**

| **Service** | **Event** | **Webhook Action** |
| --- | --- | --- |
| GitHub | New commit pushed | Notify Jenkins to start a build |
| Stripe | Payment completed | Update your billing system |
| Jira | Ticket marked as "Done" | Send a message to Slack |
| Google Forms | Form submitted | Send data to your database |

**Why Use Webhooks?**

* **Real-time**: No need to keep checking — it pushes updates immediately.
* **Automatic**: Saves time by automating tasks.
* **Efficient**: No extra traffic like constant API requests.

**Webhook vs. API (Simple Comparison)**

| **Feature** | **Webhook** | **API Polling** |
| --- | --- | --- |
| Works when | An event happens | You ask repeatedly if something changed |
| Who starts | The service (pushes info) | You (pulling info) |
| Good for | Real-time updates | Periodic checking |

**A Simple Example**

Imagine you're using GitHub and Jira:

* A developer makes a new pull request in GitHub.
* GitHub sends a webhook to Jira.
* Jira receives the data and links the pull request to the matching ticket.

**Is It Safe?**

Yes, but you should:

* Use **secret tokens** or **signatures** to verify the message.
* Only allow **HTTPS** (secure connection).
* Make sure your system checks who is sending the data.

GitHub communicates with python script using webhooks, where python is required to bridge between GitHub and Jira, python can be stored in EC2 instance. In the webhook we need to provide the URL of EC2 instance of python.

**Flask and API Development: A Practical and Scalable Approach**

**1. Introduction**

In the modern software landscape, **Application Programming Interfaces (APIs)** serve as the backbone of data exchange between clients and servers. Among various tools available for building APIs, **Flask**, a Python-based micro web framework, has gained significant popularity due to its simplicity, modularity, and scalability. Flask provides developers the flexibility to build RESTful APIs from the ground up without enforcing rigid design patterns, making it ideal for both rapid prototyping and production-level services.

**2. Flask Framework: Philosophy and Architecture**

Flask follows the **"micro, but extensible"** philosophy. It comes with the bare minimum tools to build web applications but allows seamless integration with external libraries and extensions based on specific application needs.

**Creating a Flask Application Instance**

In a Flask-based web application, the **Flask instance** serves as the central object that orchestrates the application’s configuration, request handling, and routing logic. This instance is typically created at the beginning of the application and represents the web server gateway interface (WSGI) application callable.

**🔧 Instantiation**

A Flask application is initialized using the Flask class, which is imported from the flask module. The conventional syntax is:

python

from flask import Flask

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

1. Creating EC2 instance in AWS and updating the python script in the instance.

A computer screen with a message

AI-generated content may be incorrect.

The script included is:

create\_jira\_flask.py

|  |
| --- |
| from flask import Flask  import json  import requests  from requests.auth import HTTPBasicAuth  app = Flask(\_\_name\_\_)  @app.route("/createJIRA",methods=['POST'])  def createJIRA():    url = ""      API\_TOKEN = ""      auth = HTTPBasicAuth("",API\_TOKEN )    headers = {      "Accept": "application/json",      "Content-Type": "application/json"      }      payload = json.dumps( {      "fields": {        "description": {          "content": [              {              "content": [                  {                  "text": "My first jira ticket",                  "type": "text"                  }              ],              "type": "paragraph"              }          ],          "type": "doc",          "version": 1          },                "issuetype": {          "id": "10003"          },                "project": {          "key":"SCRUM"          },        "summary": "My first jira ticket",          },      "update": {}      } )      response = requests.request(      "POST",      url,      data=payload,      headers=headers,      auth=auth      )      return json.dumps(json.loads(response.text), sort\_keys=True, indent=4, separators=(",", ": "))    app.run('0.0.0.0',port=5000) |

1. Connect the instance using Mobaxterm

Step-by-Step Process

**Step 1: Connecting to a Remote Server**

Access to a remote server is established using SSH:

ssh username@server\_ip

This provides terminal access to the remote machine for running Python code and managing files.

---

**Step 2: Setting Up a Python Environment**

A virtual environment is created to isolate project dependencies:

python3 -m venv flask\_env

source flask\_env/bin/activate

flask\_env is an isolated environment for installing packages relevant to the project.

---

**Step 3: Installing Required Python Packages**

Install the necessary packages:

pip install flask requests

flask: Used to run the web server and handle HTTP requests.

requests: Enables the application to make HTTP requests to other services (e.g., Jira APIs).

To verify installed packages:

pip list

---

**Step 4: Writing the Flask Application (jira\_project.py)**

The application file involves the following:

|  |
| --- |
| from flask import Flask, request  app = Flask(\_name\_)  @app.route('/createJIRA', methods=['POST'])  def createJIRA():  # logic to create a Jira ticket  return "Jira ticket created", 200  if \_name\_ == '\_main\_':  app.run(host='0.0.0.0', port=5000)  Key Details:  @app.route(...): Defines the endpoint to handle HTTP requests.  app.run(...): Starts the web server on port 5000.  host='0.0.0.0': Allows external access to the server. |

---

**Step 5: Running the Application**

To start the application:

python3 jira\_project.py

The Flask server will output:

Running on http://0.0.0.0:5000/

---

**Step 6: Testing the Endpoint**

The endpoint can be tested with:

curl http://<server\_ip>:5000/createJIRA -X POST

Alternatively, Postman can be used to make a POST request to the same URL.

If successful, Flask logs the request:

"POST /createJIRA HTTP/1.1" 200 -

---

**3. Flask Internal Functioning**

Flask employs an internal development server powered by Werkzeug.

When the application is started, Flask initiates a lightweight web server that listens for HTTP requests.

Incoming requests (e.g., POST /createJIRA) are routed to the corresponding Python function.

Flask returns an appropriate HTTP response (e.g., 200 OK and a confirmation message).

---

**4. Use Case for Flask and REST API**

Flask enables the exposure of backend functionality (such as Jira integration) through a RESTful web service. Other applications or users can trigger backend operations by sending HTTP requests to the Flask application.

---

**5. Considerations for Public Accessibility**

To ensure the application is publicly accessible:

Open port 5000 in the server's firewall or cloud provider's security group.

Confirm that Flask is configured with host='0.0.0.0'.

For production environments, it is advisable to use a production-grade server such as Gunicorn behind Nginx. However, Flask’s built-in server is sufficient for development and testing purposes.

A screen shot of a computer

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

1. Scripts to create Jira ticket

create\_jira.py

|  |
| --- |
| 1. # This code sample uses the 'requests' library: 2. # http://docs.python-requests.org 3. import requests 4. from requests.auth import HTTPBasicAuth 5. import json 6. url = "" 7. API\_TOKEN = "" 8. auth = HTTPBasicAuth("",API\_TOKEN ) 9. headers = { 10. "Accept": "application/json", 11. "Content-Type": "application/json" 12. } 13. payload = json.dumps( { 14. "fields": { 16. "description": { 17. "content": [ 18. { 19. "content": [ 20. { 21. "text": "My first jira ticket", 22. "type": "text" 23. } 24. ], 25. "type": "paragraph" 26. } 27. ], 28. "type": "doc", 29. "version": 1 30. },   34. "issuetype": { 35. "id": "10003" 36. },   40. "project": { 41. "key":"SCRUM" 42. },  45. "summary": "My first jira ticket",  48. }, 49. "update": {} 50. } ) 51. response = requests.request( 52. "POST", 53. url, 54. data=payload, 55. headers=headers, 56. auth=auth 57. ) 58. print(json.dumps(json.loads(response.text), sort\_keys=True, indent=4, separators=(",", ": "))) |

4. Creation of the jira ticket.

A screenshot of a computer

AI-generated content may be incorrect.

1. Creation of the Webhook

A screenshot of a computer

AI-generated content may be incorrect.

1. After running the scripts and then the environment is set and if we comment /jira then the ticket will be created.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Conclusion: This is integration of GitHub and Jira where bridged by python script which is stored in EC2 instance and GitHub is connected to python script by webhooks and API is called using Flask application.