Table of content

Objective

The primary goal of this project is to implement an automated system that seamlessly generates JIRA tickets in response to legitimate issues raised on GitHub. This automation streamlines the process of issue management by swiftly translating valid concerns from the GitHub platform into actionable tasks within the JIRA ticketing system. By establishing this integration, we aim to enhance efficiency, reduce manual intervention, and facilitate a more synchronized workflow between GitHub and JIRA, ultimately ensuring prompt and accurate resolution of identified issues.

Tools/Technologies used:

1	Github	Version control using Git
2	JIRA	Project management and issue tracking tool
3	Python	Preferred for interactive use with APIs
4	Flask	Micro web framework for Python. It simplifies web development by providing tools and libraries for building web applications.
5	AWS EC2	Launched the server and used to host flask API
6	MobaXterm	User friendly terminal
7	Bash scripting	Used to interact with the server
8	VS code	Used to write python API

System Architecture:

1. **GitHub Repository:**
    - **Action Triggered:**
        - **A QA/Tester raises an issue on the GitHub repository.**
2. **Developer Review:**
    - **Action Taken:**
        - **The developer reviews the GitHub issue to determine its legitimacy.**
3. **Comment for JIRA Ticket:**
    - **Action Taken:**
        - **If the developer finds the issue genuine, they comment on the GitHub issue with "/JIRA". This action signals the intent to create a JIRA ticket.**

4. **GitHub Webhook:**
    - **Action Triggered:**
        - **GitHub Webhook is configured to trigger on new comments. It listens for comments that include "/JIRA".**
5. **Flask API on EC2 Ubuntu Instance:**
    - **Action Triggered:**
        - **The GitHub Webhook triggers a predefined Python Flask API hosted on an EC2 Ubuntu instance.**
6. **Python Flask API Processing:**
    - **Action Taken:**
        - **The Flask API receives the GitHub webhook payload and extracts relevant information, including the comment and issue details.**
        - **It verifies that the comment includes "/JIRA".**
7. **Communication with JIRA API:**
    - **Action Taken:**
        - **If the comment is valid ("/JIRA" present), the Python API communicates with the JIRA API to create a new ticket.**
8. **JIRA Ticket Creation:**
    - **Action Taken:**
        - **The JIRA API creates a new ticket with details provided by the GitHub issue and additional information as needed.**
9. **Response to GitHub:**
    - **Action Taken:**
        - **The Python Flask API responds to the GitHub Webhook with the result of the ticket creation process.**
10. **Feedback on GitHub:**
    - **Action Taken:**
        - **If the comment was valid and a JIRA ticket was created, relevant feedback is provided on the GitHub issue.**
        - **If the comment was invalid ("/JIRA" not present), a message is printed stating that the ticket will only be created if the comment includes "/JIRA".**

Step by Step Workflow:

1) API code with explanation:

```
2) from flask import Flask,request,jsonify
3) import requests
4) from requests.auth import HTTPBasicAuth
5) import json
6)
7) app = Flask(__name__)
8)
9) @app.route("/createJIRA",methods = ['POST'])
10)def CreateJira():
11)
12)    url = "https://mayankpratap.atlassian.net/rest/api/3/issue"
13)
14)    API_Token = "" #replace with your token from JIRA
15)
16)    auth = HTTPBasicAuth("mayankpratap578@gmail.com", API_Token)
17)
18)    headers = {
19)    "Accept": "application/json",
20)    "Content-Type": "application/json"
21)    }
22)
23)    payload = json.dumps( {
24)    "fields": {
25)
26)        "description": {
27)        "content": [
28)            {
29)            "content": [
30)                {
31)                "text": "Mayank's JIRA ticket",
32)                "type": "text"
33)                }
34)            ],
35)            "type": "paragraph"
36)            }
37)        ],
38)        "type": "doc",
39)        "version": 1
40)        },
41)
42)        "issuetype": {
43)        "id": "10001"
44)        },
45)
```

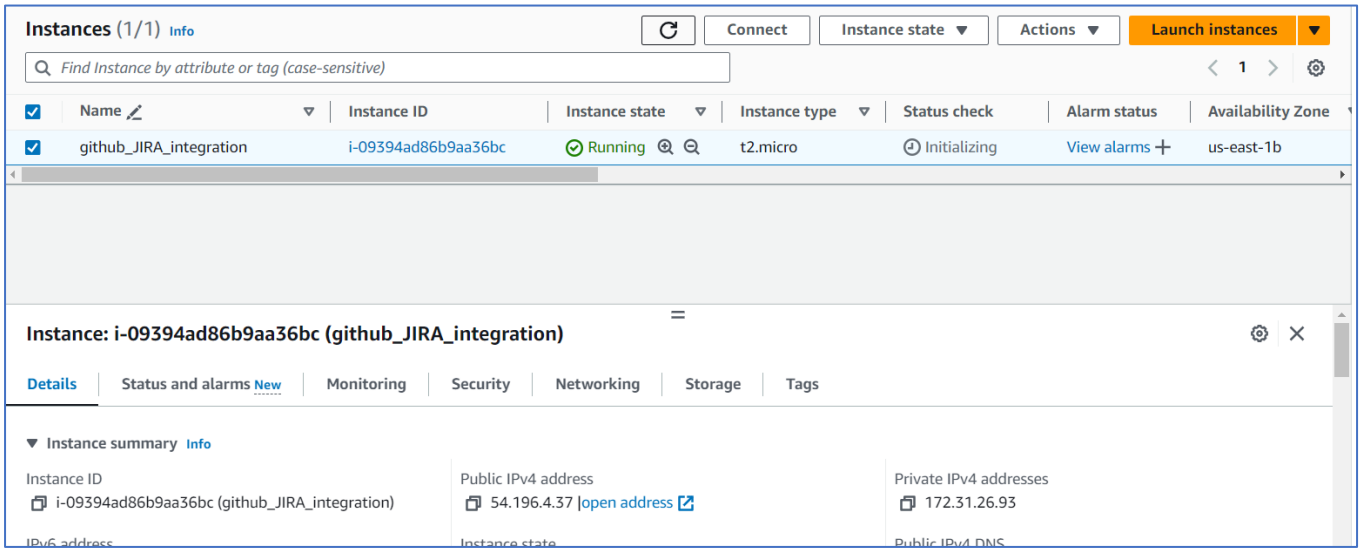
```
        "project": {
        "key": "MAYAN"
        },
        "summary": "Mayank Pratap's first JIRA ticket",


    } })

    webhook = request.json #fetching the payload
    response = None
    if webhook['comment'].get('body') == "/jira":
        response = requests.request("POST", url, data=payload,
   headers=headers, auth=auth)
        return json.dumps(json.loads(response.text), sort_keys=True,
   indent=4, separators=(",", ": "))
    else:
        print('Jira issue will be created if comment includes /jira')
        return "No action taken for this comment", 200


app.run('0.0.0.0',port=5000)
```

1. **Flask Setup:**
    - A Flask application is created.
    - The **/createJIRA** endpoint is defined to handle HTTP POST requests.
2. **JIRA Configuration:**
    - JIRA server URL is specified (**url**).
    - JIRA user's email and API token are used for authentication (**HTTPBasicAuth**).
    - Headers are set with the required content types.
3. **JIRA Ticket Payload:**
    - A sample payload is created in JSON format, containing details such as description, issue type, project key, and summary for the new JIRA ticket.
4. **Webhook Data Retrieval:**
    - The incoming JSON data from the GitHub webhook is retrieved using **request.json**.
5. **Condition Check:**
    - The code checks if the body of the comment in the GitHub webhook contains "/jira" using **webhook['comment'].get('body')**.
6. **JIRA Ticket Creation:**

- If the condition is true, a POST request is made to the JIRA server with the specified payload and headers.
- The API response from JIRA is formatted as a JSON string and returned as the API response.

7. **Fallback Message:**
   - If the condition is false (comment does not include "/jira"), a message is printed indicating that a JIRA issue will only be created if the comment includes "/jira".
8. **API Execution:**
   - The Flask application is run on **0.0.0.0** (accessible from any network interface) on port **5000**.

2) Launch an EC2 instance and SSH into it



Note – Public IP is subjective to change

3) Install PIP, flask and other dependencies with following commands:

a. Sudo apt update – to update the existing libraries

```
ubuntu@ip-172-31-89-248:~$ sudo apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:6 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1058 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [200 kB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [1237 kB]
```

b. sudo apt install python3-pip – To install pip

```
ubuntu@ip-172-31-89-248:~$ sudo apt install python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  build-essential bzip2 cpp cpp-11 dpkg-dev fakeroot fontconfig-config fonts
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl l
  libcrypt-dev libdeflate0 libdpkg-perl libexpat1-dev libfakeroot libfile-fc
  libjbig0 libjpeg-turbo8 libjpeg8 libjs-jquery libjs-sphinxdoc libjs-unders
  libstdc++-11-dev libtiff5 libtirpc-dev libtsan0 libubsan1 libwebp7 libxpm4
  python3.10-dev rpcsvc-proto zlib1g-dev
Suggested packages:
```

c. pip install flask – to install flask package

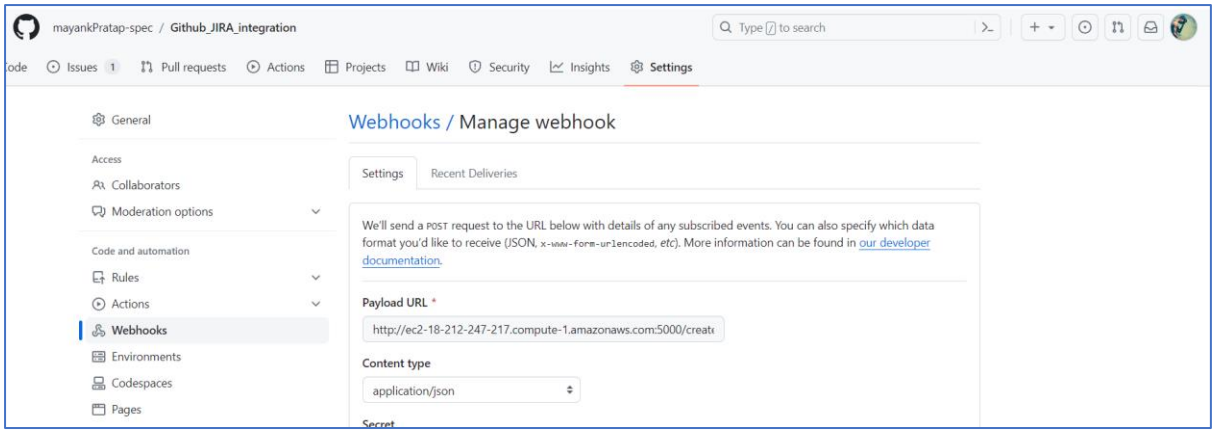
```
ubuntu@ip-172-31-89-248:~$ pip install flask
Defaulting to user installation because normal site-packages is not writeable
Collecting flask
  Downloading flask-3.0.0-py3-none-any.whl (99 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.7/99.7 KB 1.9 MB/s eta 0:00:00
Collecting Jinja2>=3.1.2
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.1/133.1 KB 6.7 MB/s eta 0:00:00
Collecting Werkzeug>=3.0.0
  Downloading werkzeug-3.0.1-py3-none-any.whl (226 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 226.7/226.7 KB 13.7 MB/s eta 0:00:00
Collecting click>=8.1.3
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 97.9/97.9 KB 14.8 MB/s eta 0:00:00
```

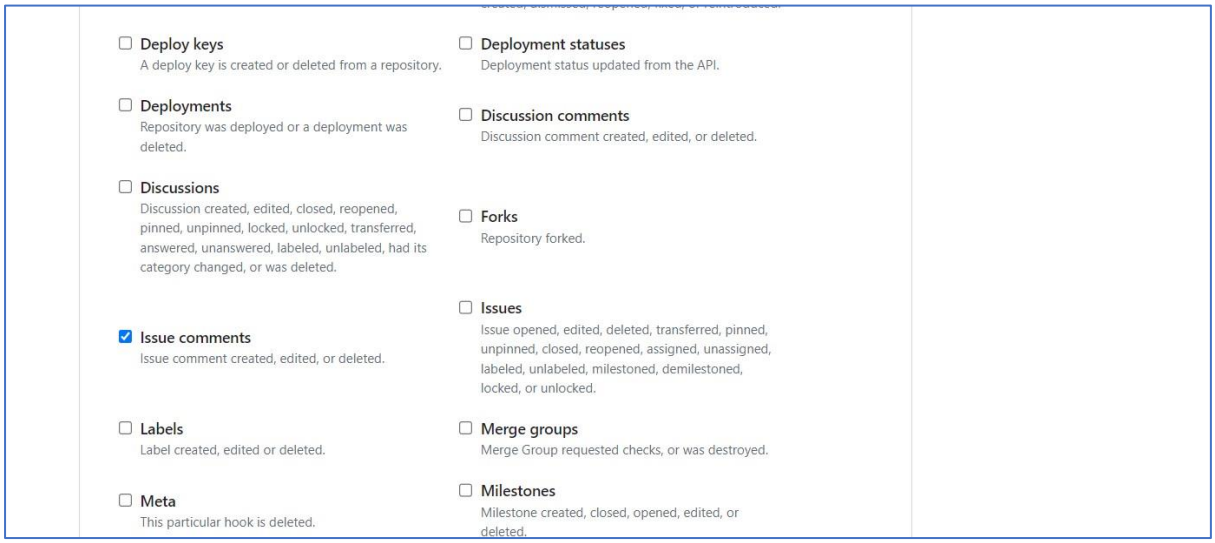
d. Using VIM text editor – Created python file which will act as flask API.

4) Setup webhook on Github repository

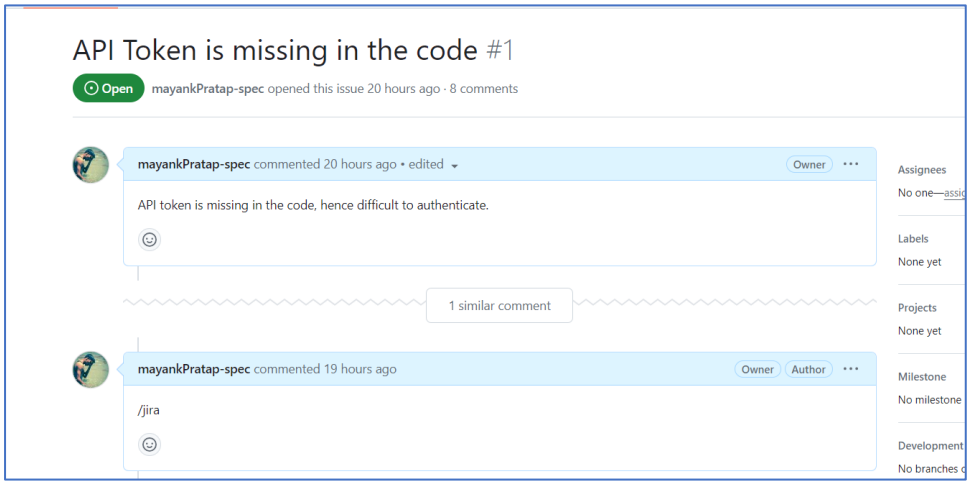
Payload URL - http://ec2-18-212-247-217.compute-1.amazonaws.com:5000/createJIRA

Created by Mayank Pratap Singh
Linked In - Mayank LinkedIn

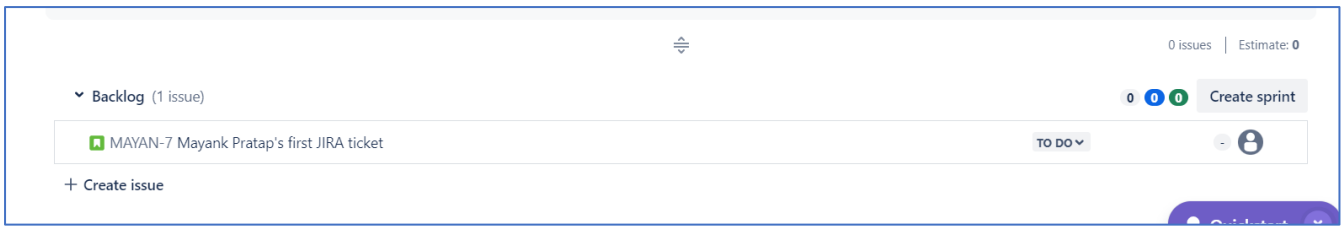




5) JIRA ticket creation on commenting /jira



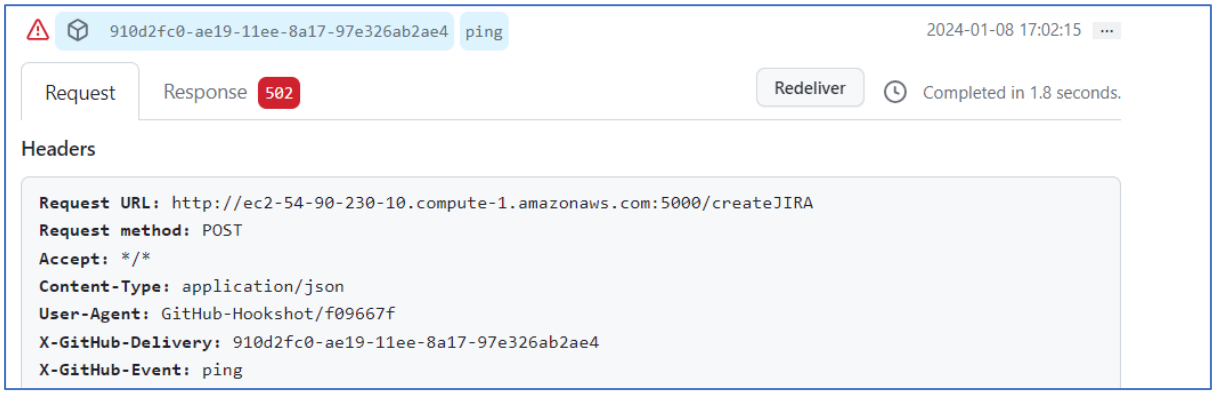
Created by Mayank Pratap Singh
Linked In - Mayank LinkedIn




```
ubuntu@ip-172-31-26-93:~$ python3 jira.py
 * Serving Flask app 'jira'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a pr
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.26.93:5000
Press CTRL+C to quit
Jira issue will be created if comment include /jira
140.82.115.113 - - [08/Jan/2024 13:21:13] "POST /createJIRA HTTP/1.1" 200 -
140.82.115.241 - - [08/Jan/2024 13:22:17] "POST /createJIRA HTTP/1.1" 200 -
Jira issue will be created if comment include /jira
140.82.115.244 - - [08/Jan/2024 13:22:46] "POST /createJIRA HTTP/1.1" 200 -
```

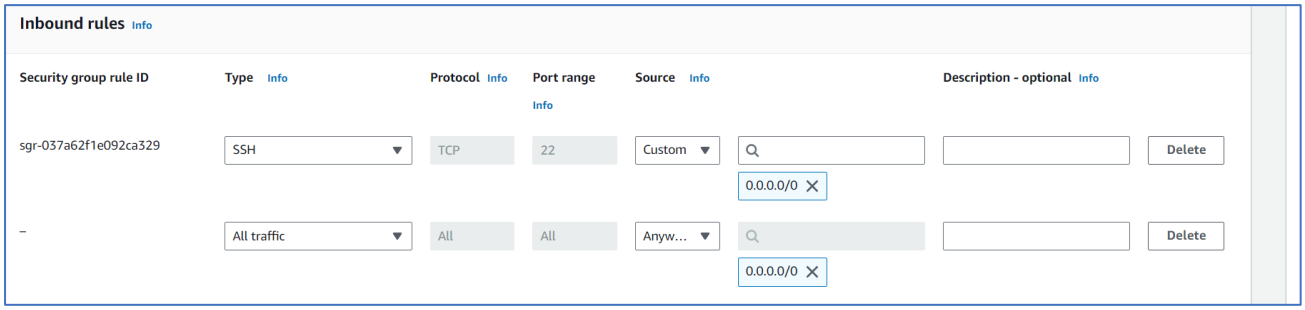
Common errors and resolution during implementation of project:

1) "Failed to connect to host" error in Github webhook



Resolution :

HTTP traffic was not allowed through security group, hence the error. Need to add a rule for the same.



2) “Port 5000 already in use” while running flask API

```
ubuntu@ip-172-31-26-93:~$ python3 github_jira.py
 * Serving Flask app 'github_jira'
 * Debug mode: off
Address already in use
Port 5000 is in use by another program. Either identify and stop that program, or start the server with a different port.
ubuntu@ip-172-31-26-93:~$
```

Resolution –

We can assign different port for the API or can kill the process if running unnecessarily as follows:

```
ubuntu@ip-172-31-26-93:~$ lsof -i:5000
COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
python3 2967 ubuntu    3u  IPv4  28101      0t0  TCP *:5000 (LISTEN)
ubuntu@ip-172-31-26-93:~$ kill 2967
ubuntu@ip-172-31-26-93:~$ lsof -i:5000
ubuntu@ip-172-31-26-93:~$
```

3) Even when comment does not included “/jira”, ticket was getting created

Resolution – implemented conditional handling

```
webhook = request.json
response = None
if webhook['comment'].get('body') == "/jira":
    response = requests.request("POST", url, data=payload, headers=headers, auth=auth)
    return json.dumps(json.loads(response.text), sort_keys=True, indent=4, separators=(",", ": "))
else:
    print('Jira issue will be created if comment includes /jira')
    return "No action taken for this comment", 200
```

References :

JIRA API docs

Github API docs