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

Service	Event	Webhook Action
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

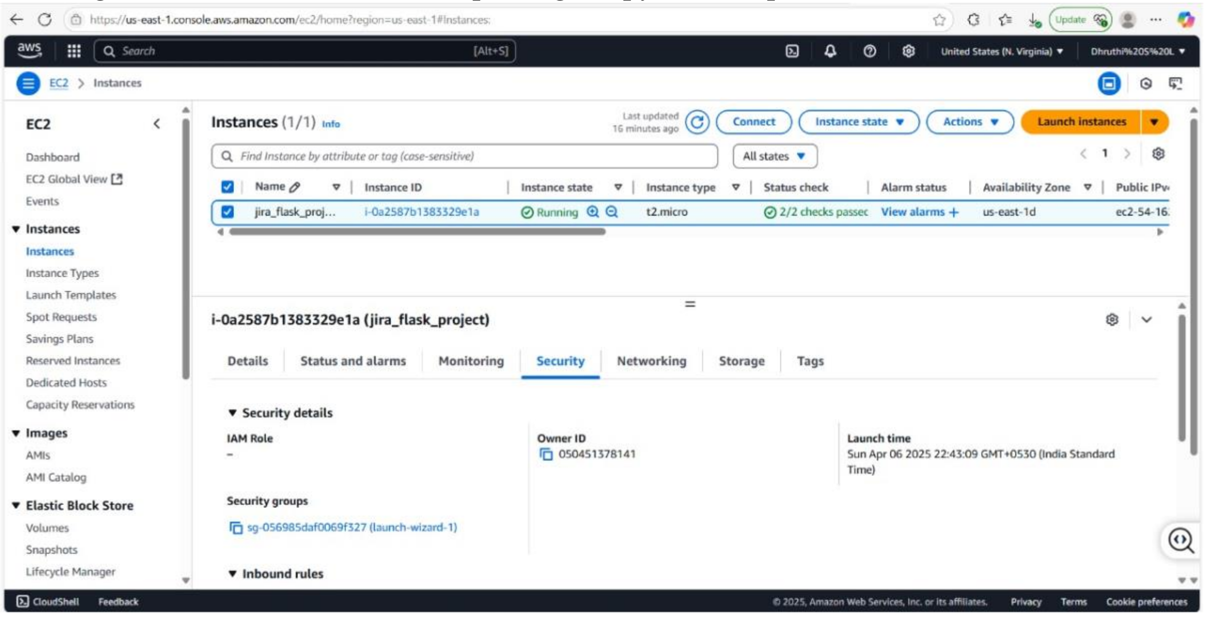
A Flask application is initialized using the Flask class, which is imported from the flask module. The conventional syntax is:

python

from flask import Flask

app = Flask(__name__)

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



https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:

Instances (1/1)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv
jira_flask_proj...	i-0a2587b1383329e1a	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1d	ec2-54-16.

i-0a2587b1383329e1a (jira_flask_project)

Security details

IAM Role: –

Owner ID: 050451378141

Launch time: Sun Apr 06 2025 22:43:09 GMT+0530 (India Standard Time)

Security groups: sg-056985daf0069f327 (launch-wizard-1)

Inbound rules

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)
```

2. 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.

---

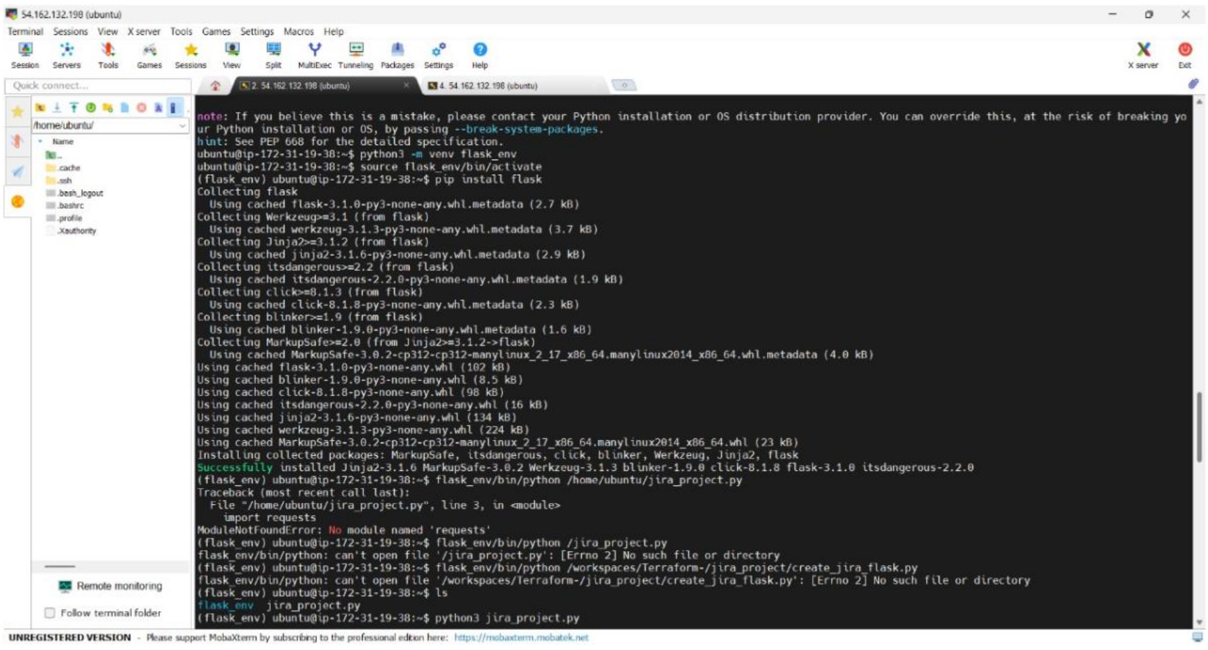
5. Considerations for Public Accessibility

To ensure the application is publicly accessible:

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

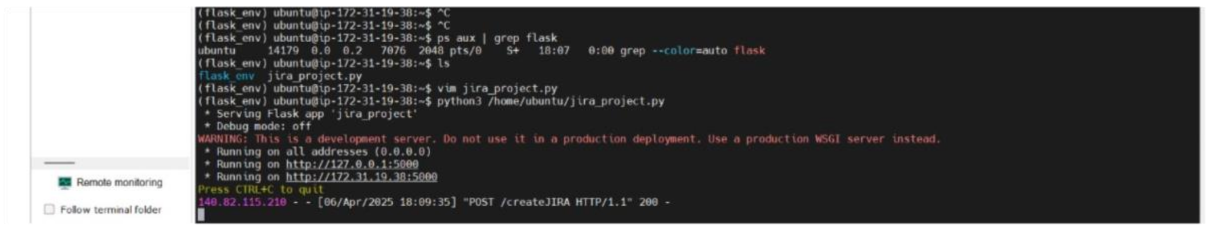
Confirm that Flask is configured with host='0.0.0.0'.

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.



```
note: If you believe this is a mistake, please contact your Python installation or OS distribution provider. You can override this, at the risk of breaking yo
ur Python installation or OS, by passing --break-system-packages.
hint: See PEP 668 for the detailed specification.
ubuntu@ip-172-31-19-38:~$ python3 -m venv flask_env
ubuntu@ip-172-31-19-38:~$ source flask_env/bin/activate
(flask_env) ubuntu@ip-172-31-19-38:~$ pip install flask
Collecting flask
  Using cached flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)
Collecting Werkzeug>=3.1 (from flask)
  Using cached werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Using cached jinja2-3.1.6-py3-none-any.whl.metadata (2.9 kB)
Collecting itsdangerous>=2.2 (from flask)
  Using cached itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.1.3 (from flask)
  Using cached click-8.1.8-py3-none-any.whl.metadata (2.3 kB)
Collecting blinker>=1.9 (from flask)
  Using cached blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
  Using cached MarkupSafe-3.0.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.0 kB)
Using cached flask-3.1.0-py3-none-any.whl (102 kB)
Using cached blinker-1.9.0-py3-none-any.whl (8.5 kB)
Using cached click-8.1.8-py3-none-any.whl (98 kB)
Using cached itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Using cached jinja2-3.1.6-py3-none-any.whl (134 kB)
Using cached werkzeug-3.1.3-py3-none-any.whl (224 kB)
Using cached MarkupSafe-3.0.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (23 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, flask
Successfully installed Jinja2-3.1.6 MarkupSafe-3.0.2 Werkzeug-3.1.3 blinker-1.9.0 click-8.1.8 flask-3.1.0 itsdangerous-2.2.0
(flask_env) ubuntu@ip-172-31-19-38:~$ flask_env/bin/python /home/ubuntu/jira_project.py
Traceback (most recent call last):
  File "/home/ubuntu/jira_project.py", line 3, in <module>
    import requests
ModuleNotFoundError: No module named 'requests'
(flask_env) ubuntu@ip-172-31-19-38:~$ flask_env/bin/python /jira_project.py
flask_env/bin/python: can't open file '/jira_project.py': [Errno 2] No such file or directory
(flask_env) ubuntu@ip-172-31-19-38:~$ flask_env/bin/python /workspaces/Terraform-/jira_project/create_jira_flask.py
flask_env/bin/python: can't open file '/workspaces/Terraform-/jira_project/create_jira_flask.py': [Errno 2] No such file or directory
(flask_env) ubuntu@ip-172-31-19-38:~$ ls
flask_env  jira_project.py
(flask_env) ubuntu@ip-172-31-19-38:~$ python3 jira_project.py
```

54.162.132.198 (ubuntu)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit
Quick connect...
2. 54.162.132.198 (ubuntu)
4. 54.162.132.198 (ubuntu)
/home/ubuntu/
Name
.cache
.ssh
.bash_logout
.bashrc
.profile
.Xauthority
Remote monitoring
Follow terminal folder
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net



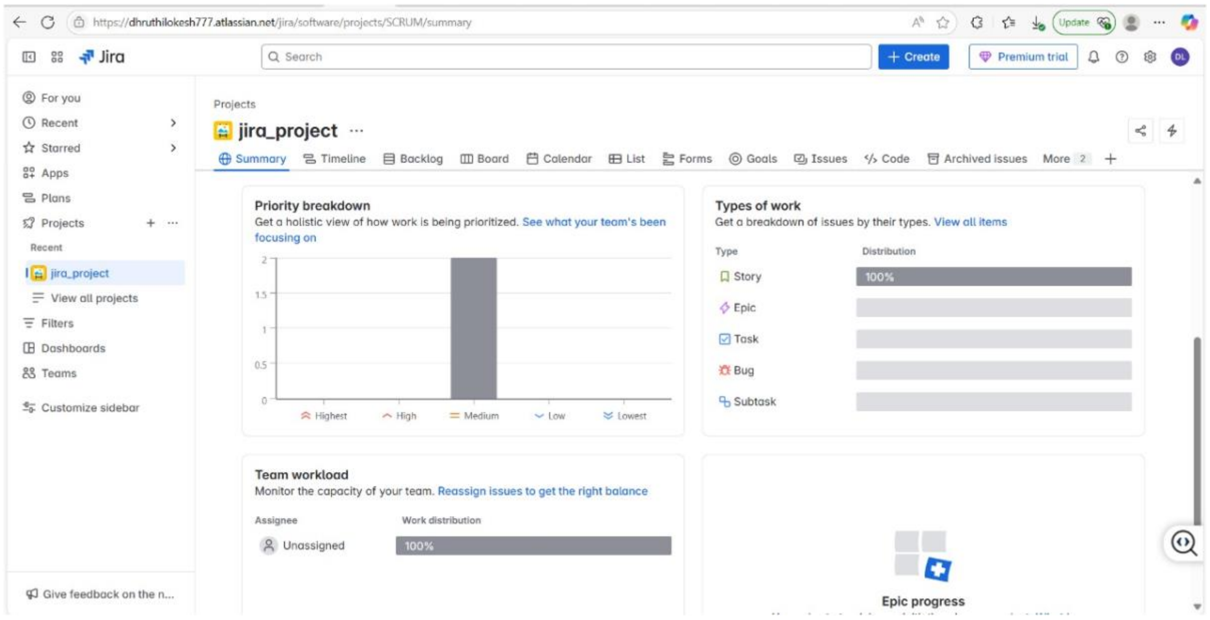
```
(flask_env) ubuntu@ip-172-31-19-38:~$ ^C
(flask_env) ubuntu@ip-172-31-19-38:~$ ^C
(flask_env) ubuntu@ip-172-31-19-38:~$ ps aux | grep flask
ubuntu     14179  0.0  0.2   7076  2048 pts/0    S+   18:07   0:00 grep --color=auto flask
(flask_env) ubuntu@ip-172-31-19-38:~$ ls
flask_env  jira_project.py
(flask_env) ubuntu@ip-172-31-19-38:~$ vim jira_project.py
(flask_env) ubuntu@ip-172-31-19-38:~$ python3 /home/ubuntu/jira_project.py
 * Serving Flask app 'jira_project'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.19.38:5000
Press CTRL+C to quit
140.82.115.210 - - [06/Apr/2025 18:09:35] "POST /createJIRA HTTP/1.1" 200 -
```

3. Scripts to create Jira ticket
   create_jira.py

```python
# This code sample uses the 'requests' library:
# http://docs.python-requests.org
import requests
from requests.auth import HTTPBasicAuth
import json

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
)

print(json.dumps(json.loads(response.text), sort_keys=True, indent=4, separators=(",", ": ")))

```

4. Creation of the jira ticket.



Priority breakdown

Get a holistic view of how work is being prioritized. See what your team's been focusing on

Types of work

Get a breakdown of issues by their types. View all items

Type	Distribution
Story	100%
Epic	
Task	
Bug	
Subtask	

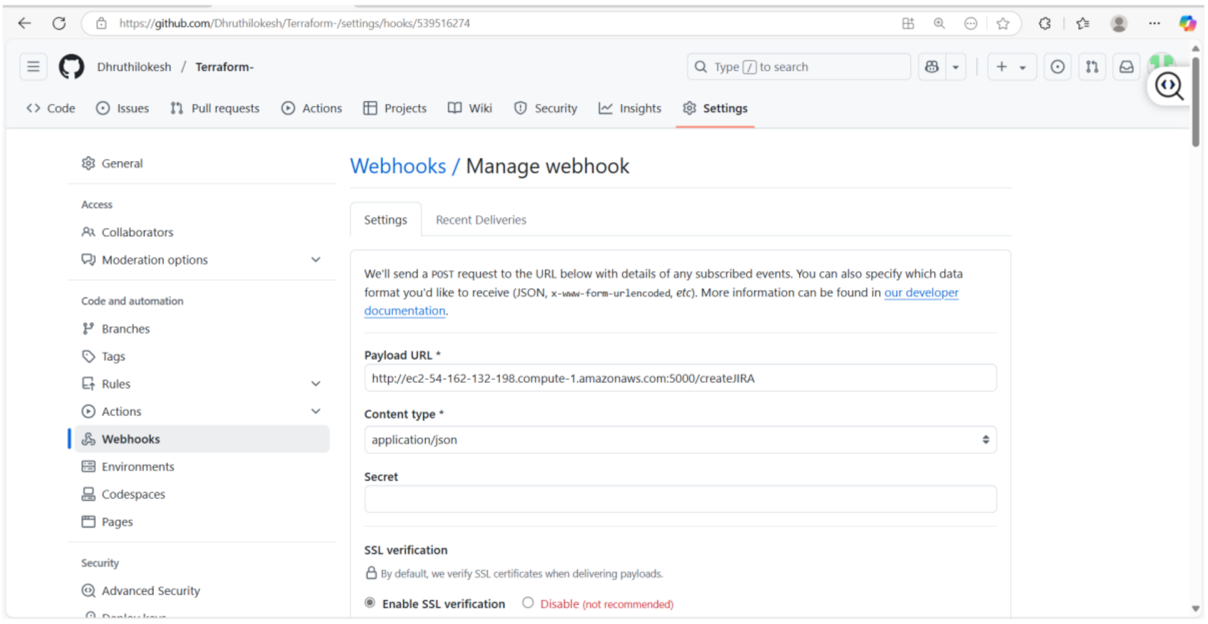
Team workload

Monitor the capacity of your team. Reassign issues to get the right balance

Assignee	Work distribution
Unassigned	100%

Epic progress

6. Creation of the Webhook



https://github.com/Dhruthilokesh/Terraform-/settings/hooks/539516274

Dhruthilokesh / Terraform-

Webhooks / Manage webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation.

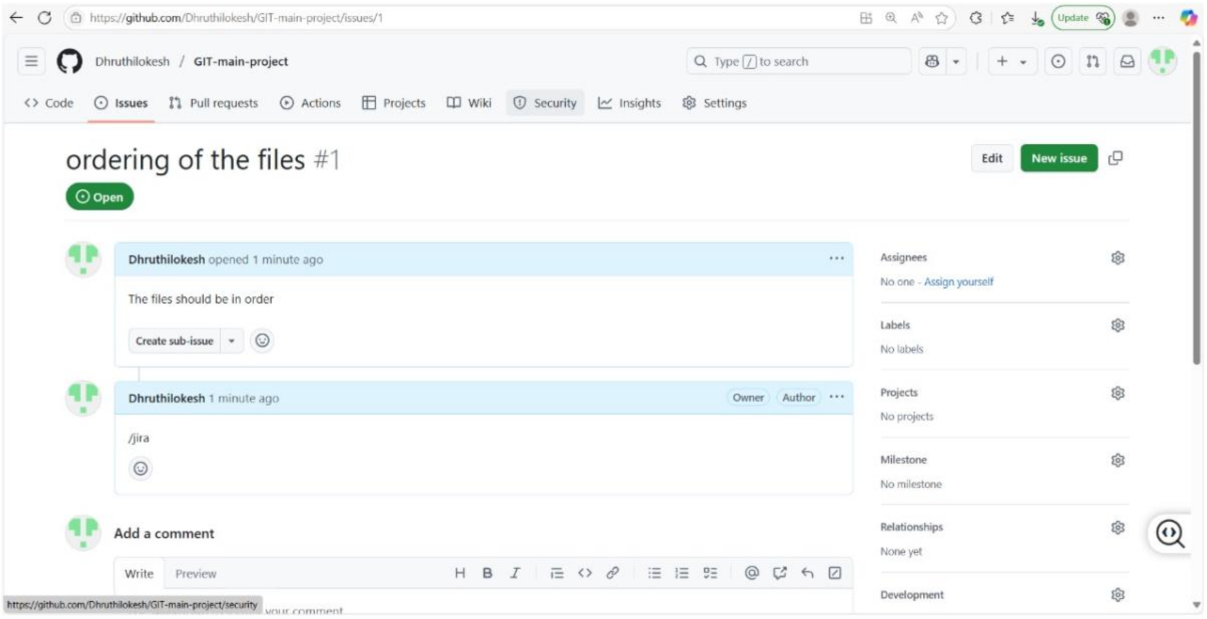
Payload URL *
http://ec2-54-162-132-198.compute-1.amazonaws.com:5000/createJIRA

Content type *
application/json

Secret

SSL verification
By default, we verify SSL certificates when delivering payloads.
[✓ Enable SSL verification] Disable (not recommended)

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



https://github.com/Dhruthilokesh/GIT-main-project/issues/1

Dhruthilokesh / GIT-main-project

ordering of the files #1

Open

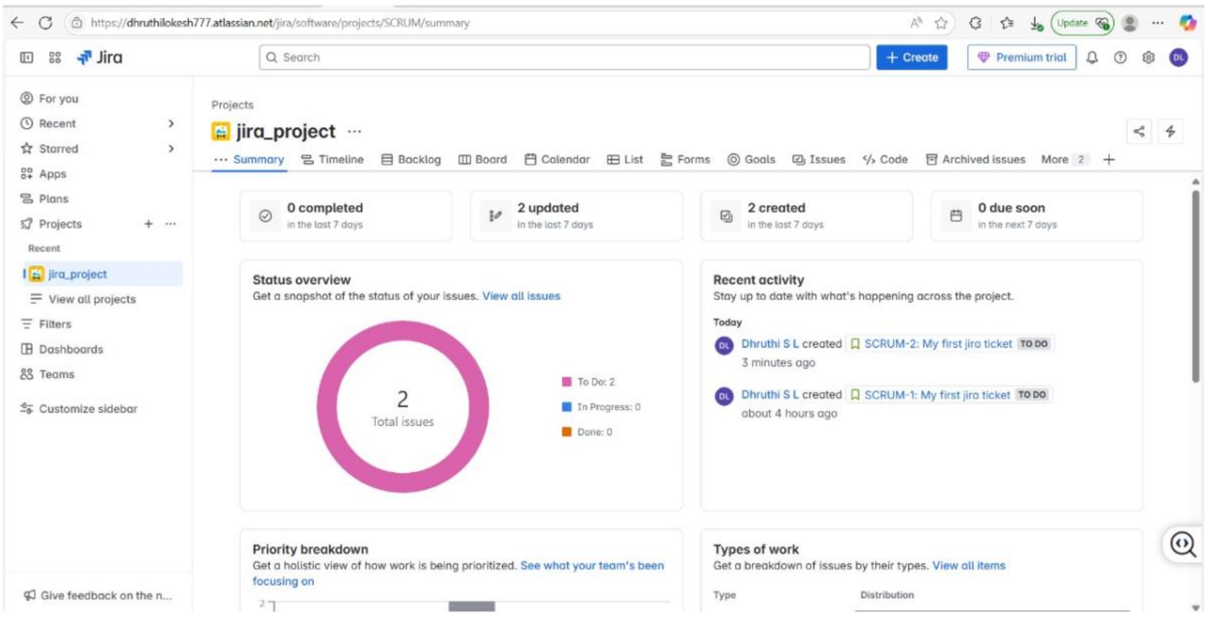
Dhruthilokesh opened 1 minute ago

The files should be in order

Dhruthilokesh 1 minute ago

/jira

Assignees: No one - Assign yourself
Labels: No labels
Projects: No projects
Milestone: No milestone
Relationships: None yet


https://dhruthilokesh777.atlassian.net/jira/software/projects/SCRUM/summary

Projects

jira_project

0 completed	in the last 7 days
2 updated	in the last 7 days
2 created	in the last 7 days
0 due soon	in the next 7 days

Status overview
Get a snapshot of the status of your issues. View all issues

2 Total issues

- To Do: 2
- In Progress: 0
- Done: 0

Recent activity
Stay up to date with what's happening across the project.

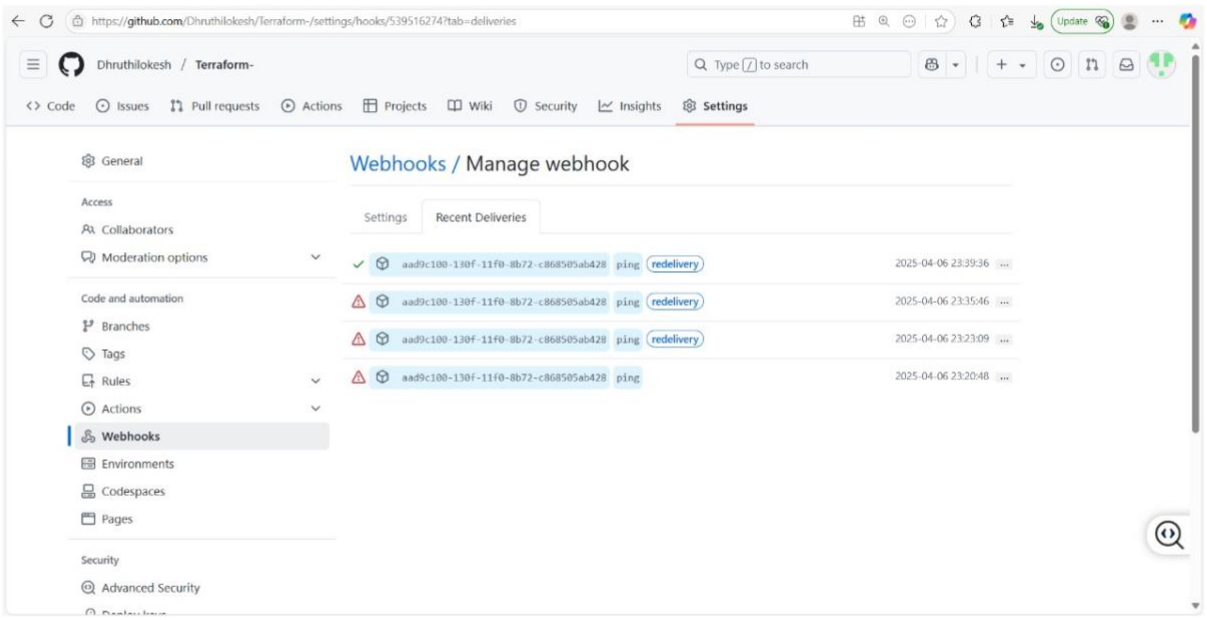
Today

Dhruthi S L created SCRUM-2: My first jira ticket TO DO
3 minutes ago

Dhruthi S L created SCRUM-1: My first jira ticket TO DO
about 4 hours ago

Priority breakdown
Get a holistic view of how work is being prioritized. See what your team's been focusing on

Types of work
Get a breakdown of issues by their types. View all items

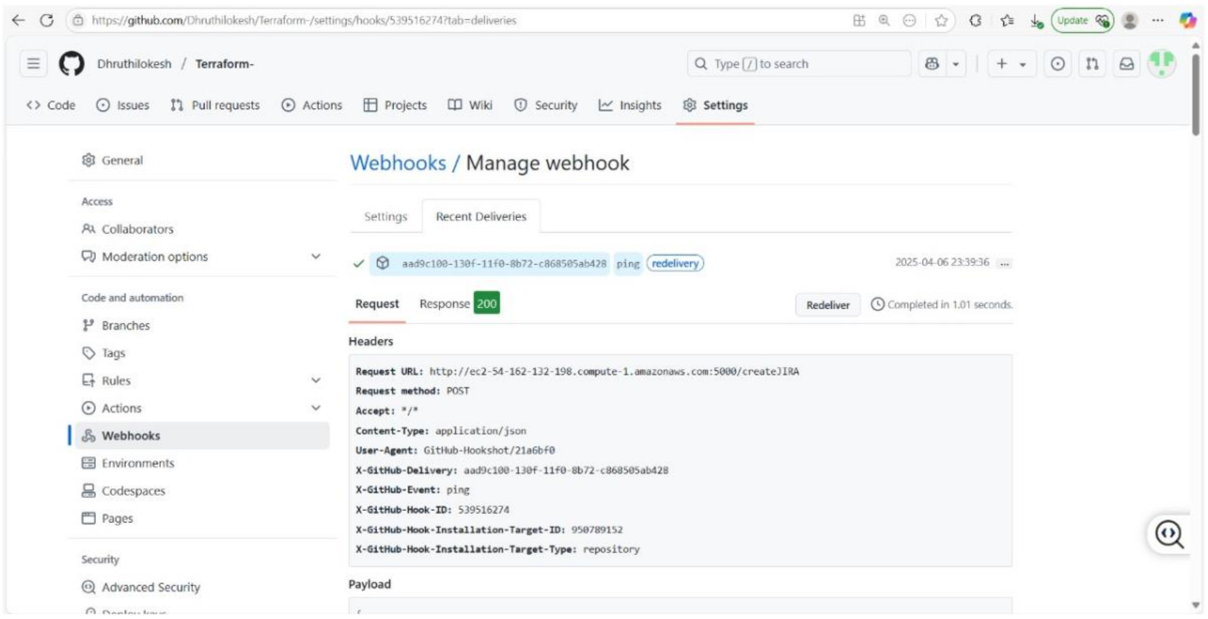

https://github.com/Dhruthilokesh/Terraform-/settings/hooks/539516274?tab=deliveries

Dhruthilokesh / Terraform-

Webhooks / Manage webhook

Settings | Recent Deliveries

```
✓  aad9c100-130f-11f0-8b72-c868505ab428  ping  redelivery  2025-04-06 23:39:36
⚠  aad9c100-130f-11f0-8b72-c868505ab428  ping  redelivery  2025-04-06 23:35:46
⚠  aad9c100-130f-11f0-8b72-c868505ab428  ping  redelivery  2025-04-06 23:23:09
⚠  aad9c100-130f-11f0-8b72-c868505ab428  ping              2025-04-06 23:20:48
```


https://github.com/Dhruthilokesh/Terraform-/settings/hooks/539516274?tab=deliveries
Dhruthilokesh / Terraform-
Code Issues Pull requests Actions Projects Wiki Security Insights Settings
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Advanced Security

Webhooks / Manage webhook

Settings | Recent Deliveries

✓ aad9c100-130f-11f0-8b72-c868505ab428 ping redelivery 2025-04-06 23:39:36

Request | Response 200 | Redeliver | Completed in 1.01 seconds.

Headers

```
Request URL: http://ec2-54-162-132-198.compute-1.amazonaws.com:5000/createJIRA
Request method: POST
Accept: */*
Content-Type: application/json
User-Agent: GitHub-Hookshot/21a6bf0
X-GitHub-Delivery: aad9c100-130f-11f0-8b72-c868505ab428
X-GitHub-Event: ping
X-GitHub-Hook-ID: 539516274
X-GitHub-Hook-Installation-Target-ID: 950789152
X-GitHub-Hook-Installation-Target-Type: repository
```

Payload

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.