Project Summary

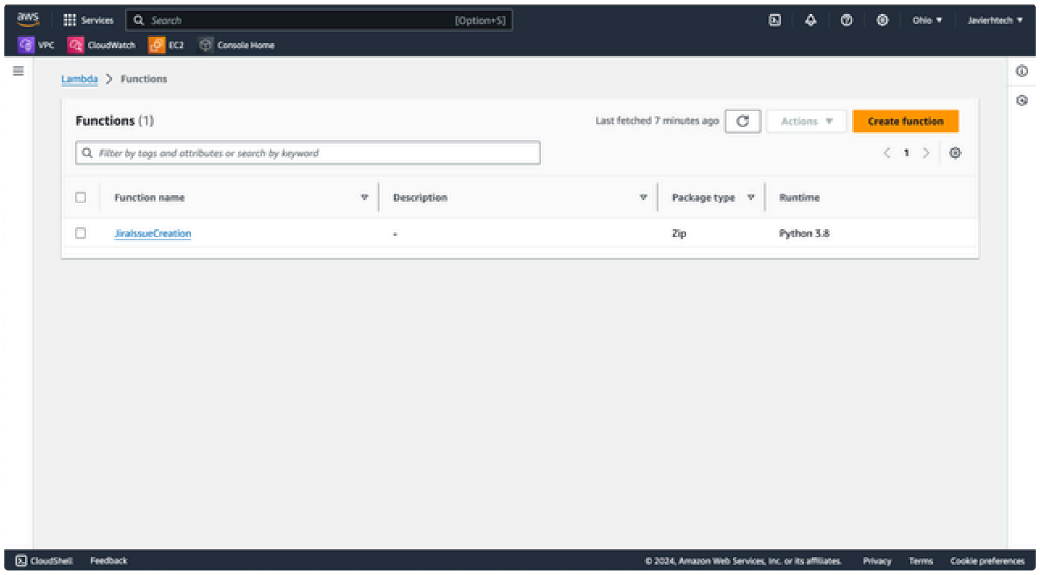
Today, I'll walk you through a project where I integrated AWS Lambda with Jira to automate the creation of Jira issues based on specific events. This integration significantly improves task management by ensuring that events triggering AWS Lambda functions can automatically generate corresponding Jira tasks. This not only streamlines project workflows but also enhances efficiency in managing and tracking tasks.

Detailed Walkthrough

1. **Setting Up the AWS Environment**
   - **Create a New AWS Lambda Function:**

     ```
     Navigate to the AWS Lambda console. Click on "Create function." Choose "Author from scratch," give it a name
     (e.g., `JiraIssueCreation`), and select the Python 3.12 runtime. Click "Create function."
     ```

   This step sets up the Lambda function where we'll run our code. AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers.



Function name	Description	Package type	Runtime
JiraIssueCreation	-	Zip	Python 3.8

2. **Preparing the Deployment Package**
   - **Create a Directory for Dependencies:**

     Code

     ```
     mkdir lambda_package cd lambda_package
     ```

   Here, I'm creating a directory named `lambda_package` to hold the necessary dependencies and the Lambda function code.
   - **Install Required Packages:**

     Code

     ```
     pip3 install requests==2.28.2 -t . pip3 install urllib3==1.26.15 -t .
     ```

   These commands use `pip3` to install the `requests` and `urllib3` packages into the current directory (`-t .`). `pip3` is a package installer for Python, and it's used to install and manage software packages written in Python. The `-t .` option specifies the target directory where these packages will be installed, which in this case is the current directory. This ensures that the dependencies are packaged together with the Lambda function code.

- **Add the Lambda Function Code:**
  - Save the following script as `lambda_function.py` in the `lambda_package` directory:

```
import json
import requests
import base64

JIRA_URL = "Input Your Project URL here"
JIRA_USER = "Input Your JIRA User"
JIRA_API_TOKEN =   "" # Replace with the new token
JIRA_PROJECT_KEY = "" Replace with Project Key

# Base64 encode the API token
auth_token = base64.b64encode(f"{JIRA_USER}:{JIRA_API_TOKEN}".encode()).decode()

def lambda_handler(event, context):
    issue_summary = event.get('summary', 'Default Summary')
    issue_description = event.get('description', 'Default Description')

    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Basic {auth_token}"
    }

    issue_data = {
        "fields": {
            "project": {
                "key": JIRA_PROJECT_KEY
            },
            "summary": issue_summary,
            "description": issue_description,
            "issuetype": {
                "name": "Task"
            }
        }
    }

    response = requests.post(
        f"{JIRA_URL}/rest/api/2/issue",
        headers=headers,
        data=json.dumps(issue_data)
    )

    try:
        response_data = response.json()
    except json.JSONDecodeError:
        response_data = response.text

    print("Request Headers:", headers)
    print("Request Data:", issue_data)
    print("Response Status Code:", response.status_code)
    print("Response Data:", response_data)

    return {
        "statusCode": response.status_code,
        "body": response_data
    }

```

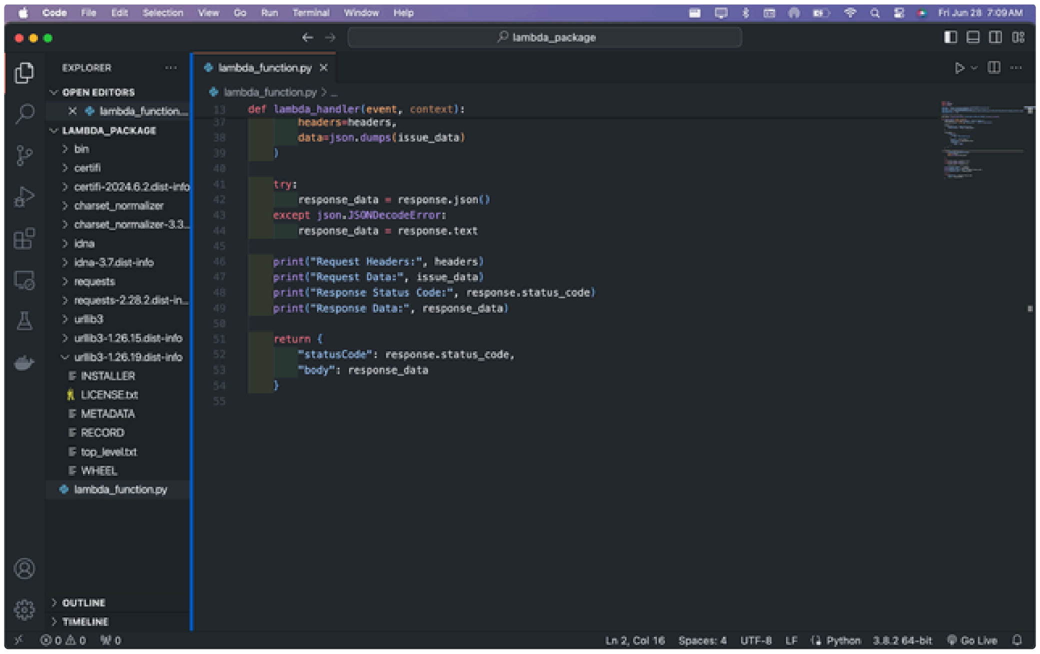
This script defines the Lambda function that will create issues in Jira using the Jira REST API. The function constructs the request with the necessary headers and payload, then sends it to the Jira API endpoint.

- **Create a ZIP Package:**

  Code

  `zip -r lambda_package.zip .`

This command creates a ZIP file (`lambda_package.zip`) containing all the files in the current directory. This ZIP file will be uploaded to AWS Lambda.

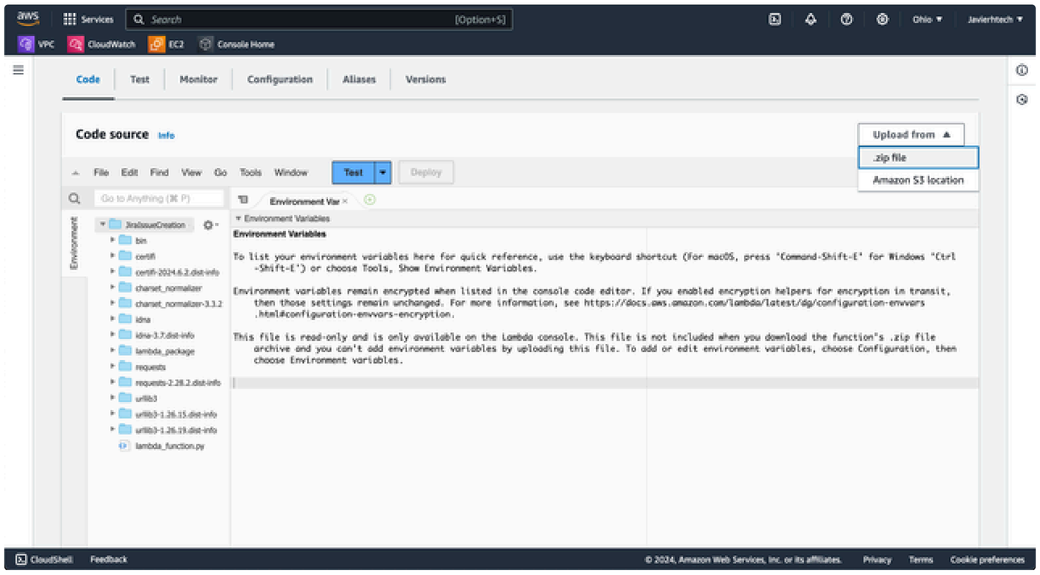


```python
def lambda_handler(event, context):
            headers=headers,
            data=json.dumps(issue_data)
        )

        try:
            response_data = response.json()
        except json.JSONDecodeError:
            response_data = response.text

        print("Request Headers:", headers)
        print("Request Data:", issue_data)
        print("Response Status Code:", response.status_code)
        print("Response Data:", response_data)

        return {
            "statusCode": response.status_code,
            "body": response_data
        }

```

3. **Uploading the Deployment Package**
   - **Upload to AWS Lambda:**

     `In the AWS Lambda console, select your function (`JiraIssueCreation`). Under "Code source," click "Upload from" and select ".zip file." Upload the `lambda_package.zip` file.`

   By uploading the ZIP file, I'm deploying the Lambda function code and its dependencies to AWS Lambda.



```
Environment Variables

To list your environment variables here for quick reference, use the keyboard shortcut (For macOS, press 'Command-Shift-E' for Windows 'Ctrl
    -Shift-E') or choose Tools, Show Environment Variables.

Environment variables remain encrypted when listed in the console code editor. If you enabled encryption helpers for encryption in transit,
    then those settings remain unchanged. For more information, see https://docs.aws.amazon.com/lambda/latest/dg/configuration-envvars
    .html#configuration-envvars-encryption.

This file is read-only and is only available on the Lambda console. This file is not included when you download the function's .zip file
    archive and you can't add environment variables by uploading this file. To add or edit environment variables, choose Configuration, then
    choose Environment variables.
```

4. **Configuring the Lambda Function**
    - **Set Up Environment Variables:**

      `In the Lambda function configuration, add environment variables for `JIRA_URL`, `JIRA_USER`, `JIRA_API_TOKEN`, and `JIRA_PROJECT_KEY`.`

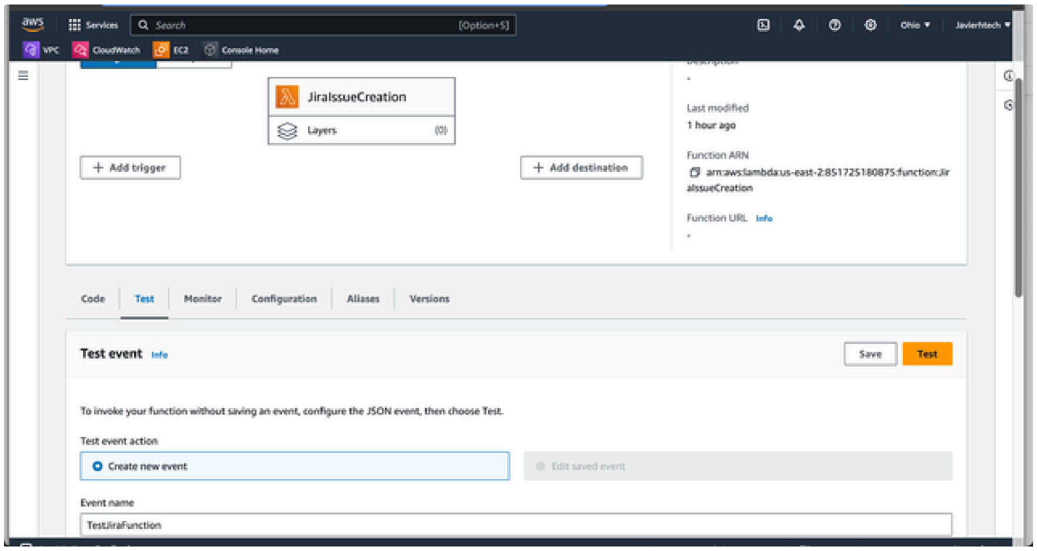
   These environment variables store the Jira instance URL, user email, API token, and project key. They allow the function to authenticate and interact with the Jira API securely.

```
JIRA_URL = "" #Your Jira URL
JIRA_USER = "youremail@email.com" #Your Jira USER Email
JIRA_API_TOKEN = ""  # Replace with the new token
JIRA_PROJECT_KEY = "" #Your Jira Project Key
```

5. **Testing the Lambda Function**
    - **Create a Test Event:**

      `In the AWS Lambda console, go to the "Test" tab. Create a new test event with the following JSON: { "summary": "Test Issue from AWS Lambda", "description": "This issue was created by testing the AWS Lambda function." } Save and run the test event. Verify the logs and check if the issue is created in your Jira project.`

   Creating a test event allows me to simulate a trigger for the Lambda function. After running the test, I verify the logs to ensure the function executed correctly and the issue was created in Jira.



Last modified
1 hour ago

Function ARN
arn:aws:lambda:us-east-2:851725180875:function:JiraIssueCreation

Function URL
-

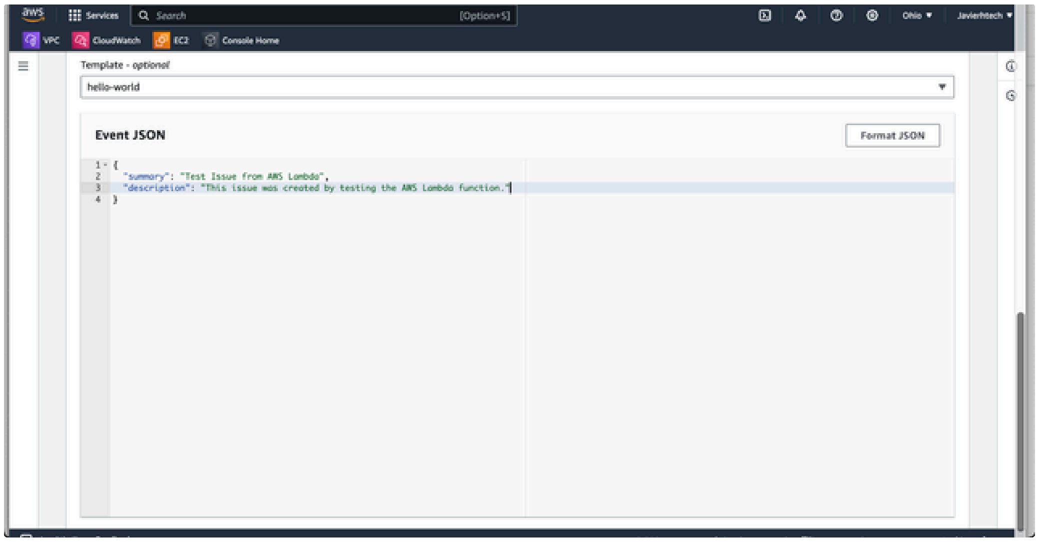
Test event

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action
Create new event
Edit saved event

Event name
TestJiraFunction

Test Tab

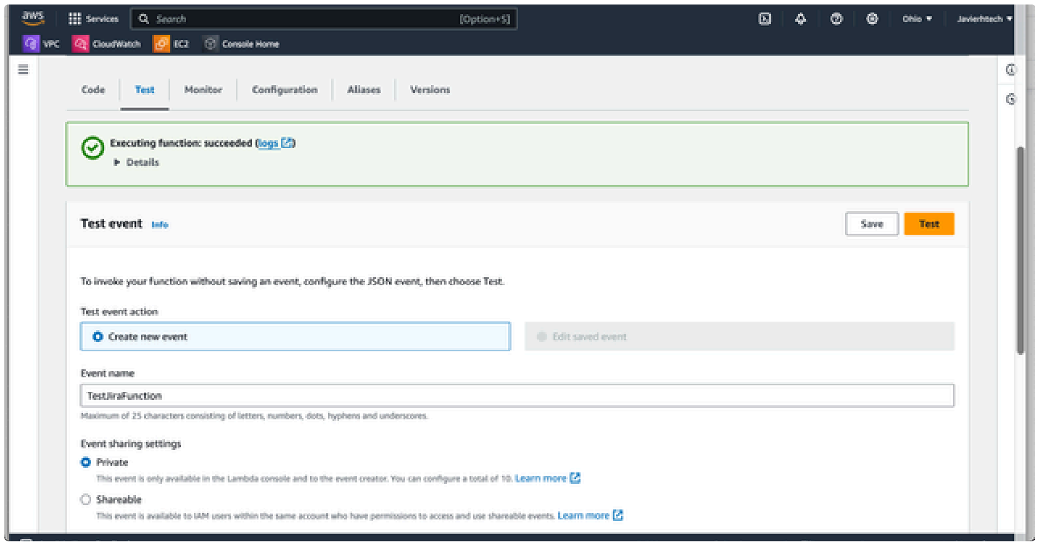


Template - *optional*
hello-world

Event JSON

```
{
  "summary": "Test Issue from AWS Lambda",
  "description": "This issue was created by testing the AWS Lambda function."
}
```

JSON For Test Event


Executing function: succeeded (logs)

Details

Test event Info

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

● Create new event ○ Edit saved event

Event name

TestJiraFunction

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

● Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more

○ Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Test Succeeded

6. **Verifying the Issue in Jira**
   - **Check Jira for New Issues:**

     ```
     Log in to your Jira account. Navigate to your project dashboard. Check the "Issues" section for a new issue
     with the summary "Test Issue from AWS Lambda."
     ```

   I log into Jira and check the project dashboard to confirm that the new issue was created successfully.

Type	Key	Summary	Assignee	Priority	Status	Updated
✓	JAV-6	Test Issue from AWS Lambda	Unassigned	=	TO DO	Jun 28, 2024, 11:16
✓	JAV-5	Test Issue from AWS Lambda	Unassigned	=	TO DO	Jun 28, 2024, 10:08
✓	JAV-4	Test Issue from AWS Lambda	Unassigned	=	TO DO	Jun 28, 2024, 10:08
🟩	JAV-3	Integrate AWS services with the on-premises infrastructu...	JH J H	=	TO DO	Jun 28, 2024, 09:42
🟥	JAV-2	Investigate and resolve network connectivity issues.	Unassigned	=	TO DO	Jun 28, 2024, 09:42
✓	JAV-1	Set up new user accounts in Active Directory	JH J H	=	TO DO	Jun 28, 2024, 09:42

6 items   Synced just now

Summary

In this project, I integrated AWS Lambda with Jira to automate the creation of Jira issues based on specific events. Here's what I accomplished:

1. **AWS Environment Setup:**
   - Created a new AWS Lambda function using Python 3.8.
   - Prepared a deployment package with necessary dependencies.
2. **Lambda Function Code:**
   - Wrote a function to interact with the Jira REST API, creating issues based on event data.
   - Configured environment variables for secure and flexible API interaction.
3. **Testing and Verification:**

- Created and executed a test event to simulate issue creation.
- Verified successful issue creation in Jira by checking the project dashboard.

This integration showcases my ability to automate processes, manage cloud services, and efficiently handle project management tools. These skills are crucial for roles where automation and cloud service management are essential. This project highlights my proficiency in using AWS services, API integration, and improving task management workflows.