Yash Mutatkar **Date: Nov 6 2024**

PHP Application Monitoring with AWS CloudWatch & GitHub Actions

Introduction

In this project, I set up an automated monitoring system for my PHP application hosted on an EC2 instance using **AWS CloudWatch** and **GitHub Actions**.

What is AWS CloudWatch?

AWS CloudWatch is a monitoring service for AWS cloud resources and applications. It allows you to track metrics like CPU usage, memory usage, disk I/O, and network traffic in real time. In addition to monitoring system health, CloudWatch allows you to send custom metrics, create alarms, and visualize your application performance with dashboards.

For this project, I used **CloudWatch** to:

- Monitor application performance metrics like **response time** and **request count**.
- Monitor the system's health, such as **CPU usage** and **memory usage**.

What is GitHub Actions?

GitHub Actions is a powerful tool for automating workflows in your GitHub repository. With GitHub Actions, you can automate tasks like deployments, running tests, building applications, and more. It works directly from your repository and can be configured to trigger workflows on specific events such as code pushes, pull requests, or even on a schedule (cron).

For my project, **GitHub Actions** runs a scheduled workflow every 5 minutes that:

- Checks the health of the PHP application.
- Sends important metrics to **AWS CloudWatch**.
- Creates issues on GitHub if something goes wrong.

Steps to Set Up Application Monitoring

Step 1: Setting Up the PHP Application

I began by creating a simple PHP application that I hosted on an **EC2 instance** with **Apache**.

The application has three sections:

1. **Hello World** with a dynamic color wave effect on the name "Yash Mutatakar".
2. A **basic calculator** that performs simple arithmetic operations.
3. A **dice roll game** where the user guesses a number between 1 and 6 and tries to match it with a randomly rolled dice value.

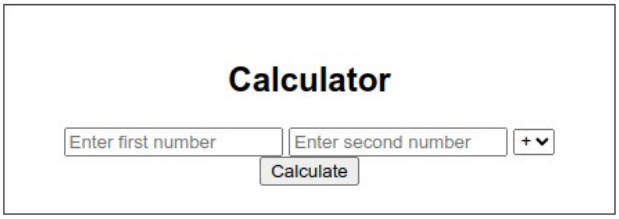
This application is accessible through a URL:
`http://<EC2_IP_ADDRESS>/app.php`

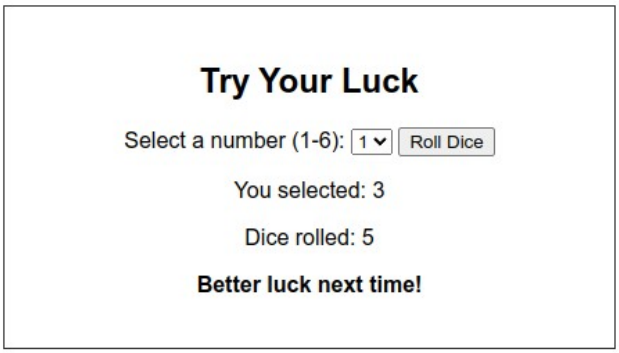
Step 2: Installing Apache and PHP on EC2

To run the PHP application, I installed Apache and PHP on my EC2 instance:

1. I connected to the EC2 instance via SSH.
2. Installed Apache: `sudo apt-get install apache2`
3. Installed PHP: `sudo apt-get install php libapache2-mod-php`
4. Uploaded my PHP application to `/var/www/html` on the EC2 instance.

After setting everything up, my application was live and accessible from the browser.





Step 3: Writing a Bash Script for CloudWatch Metrics

Next, I wrote a Bash script that would collect metrics from the application and send them to AWS CloudWatch. This script runs on the EC2 instance and performs the following actions:

1. **Measures Response Time**: It makes an HTTP request to my PHP application and measures how long it takes to get a response (in milliseconds).
2. **Tracks Request Count and Errors**: It tracks how many requests the app is receiving, and if the app returns any errors (status codes 500 or above), it sends an error count metric.

3. **Collects System Metrics**: It gathers **CPU usage** and **memory usage** using system commands (`top` and `free`) and sends these metrics to CloudWatch.

monitor.yml M × app.php $ phpappmonitor.sh

.github > workflows > monitor.yml

```yaml
name: PHP Application Monitoring

on:
  schedule:
    - cron: '*/5 * * * *'
  workflow_dispatch:

permissions:
  contents: read
  issues: write

env:
  AWS_REGION: us-east-1
  APP_URL: http://3.81.166.101/app.php
  INSTANCE_ID: i-009abcc462d996efd

jobs:
  monitor:
    runs-on: ubuntu-latest

    steps:
    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v1
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ env.AWS_REGION }}

    - name: Check application and send metrics
      id: health_check
      run: |
        # Check application health
        start_time=$(date +%s%N)
        http_code=$(curl -o /dev/null -s -w "%{http_code}" ${{ env.APP_URL }})
        end_time=$(date +%s%N)
        response_time=$(( ($end_time - $start_time) / 1000000 ))

        echo "HTTP Status Code: $http_code"
        echo "Response Time: $response_time ms"

        # Send metrics to CloudWatch
        aws cloudwatch put-metric-data \
          --namespace "PHP_Application" \
          --metric-name "ResponseTime" \
          --value $response_time \
          --unit Milliseconds \
          --dimensions Instance=${{ env.INSTANCE_ID }}
```

```
        # Record availability (1 for up, 0 for down)
        aws cloudwatch put-metric-data \
          --namespace "PHP_Application" \
          --metric-name "Availability" \
          --value $([ "$http_code" -eq 200 ] && echo 1 || echo 0) \
          --unit Count \
          --dimensions Instance=${{ env.INSTANCE_ID }}

    - name: Get CPU metrics
      run: |
        # Calculate timestamps
        end_time=$(date -u +"%Y-%m-%dT%H:%M:%SZ")
        start_time=$(date -u -d "5 minutes ago" +"%Y-%m-%dT%H:%M:%SZ")

        # Get CPU metrics
        aws cloudwatch get-metric-statistics \
          --namespace AWS/EC2 \
          --metric-name CPUUtilization \
          --dimensions Name=InstanceId,Value=${{ env.INSTANCE_ID }} \
          --start-time "$start_time" \
          --end-time "$end_time" \
          --period 300 \
          --statistics Average

    - name: Create issue on failure
      if: failure()
      run: |
        gh auth login --with-token <<< "${{ secrets.GITHUB_TOKEN }}"
        gh issue create \
          --title "Application Monitoring Alert" \
          --body "The application monitoring check has failed. Please investigate." \
          --repo "${{ github.repository }}"
```

Step 4: Setting Up AWS CloudWatch
I set up CloudWatch to accept custom metrics and display them in the CloudWatch console:

1. **Custom Namespace**: I created a custom namespace for my metrics, `CustomAppMetrics`.
2. **Metrics**: I sent the following metrics:
   - **Response Time** (in milliseconds)
   - **Request Count** (number of requests received)
   - **Error Count** (number of errors occurred)
   - **CPU Usage** (percentage of CPU utilization)
   - **Memory Usage** (percentage of memory used)

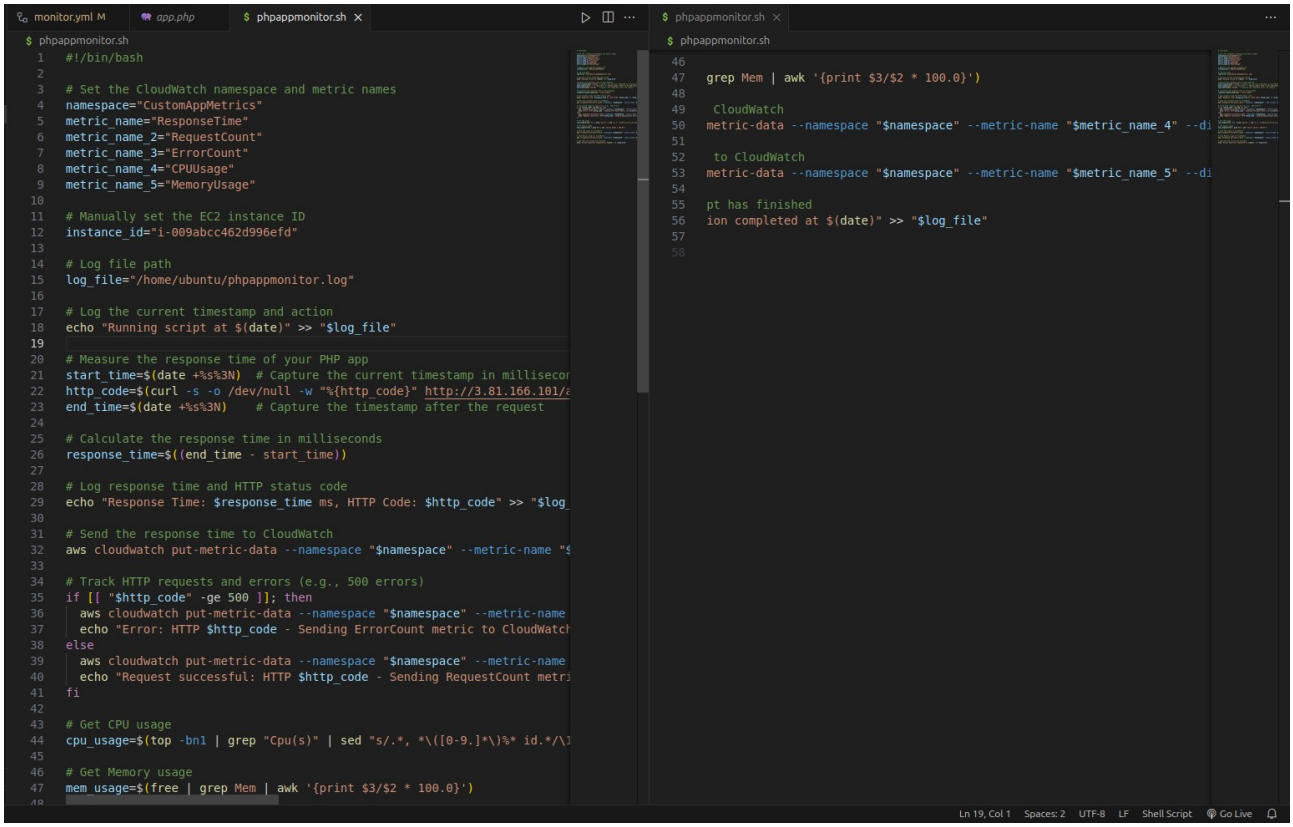
These metrics were stored under the **CustomAppMetrics** namespace in CloudWatch.

Step 5: Automating Monitoring with GitHub Actions

To automate the monitoring process, I used **GitHub Actions**. Here's how I set it up:

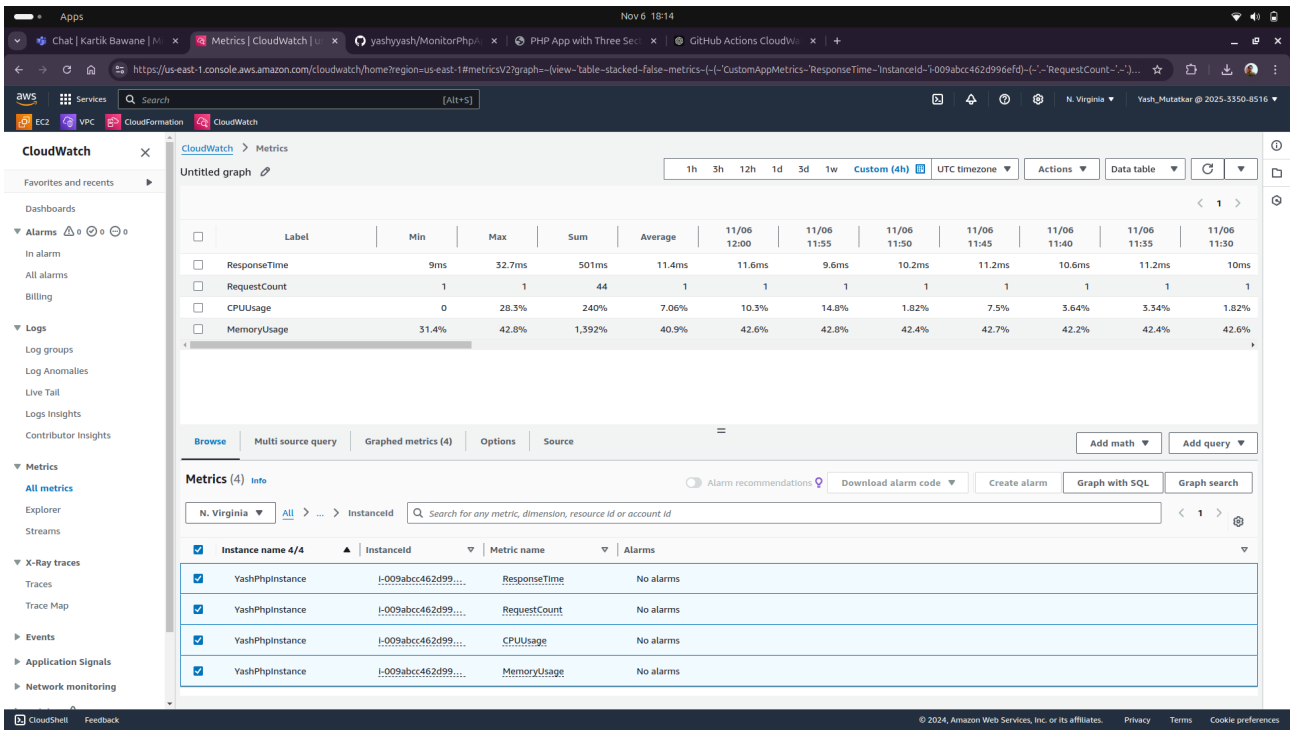
1. **Create a GitHub Actions Workflow File**: In my GitHub repository, I created a workflow file `.github/workflows/monitor.yml`.
2. **Schedule the Workflow**: The workflow runs on a schedule (every 5 minutes) using the cron syntax `*/5 * * * *`.

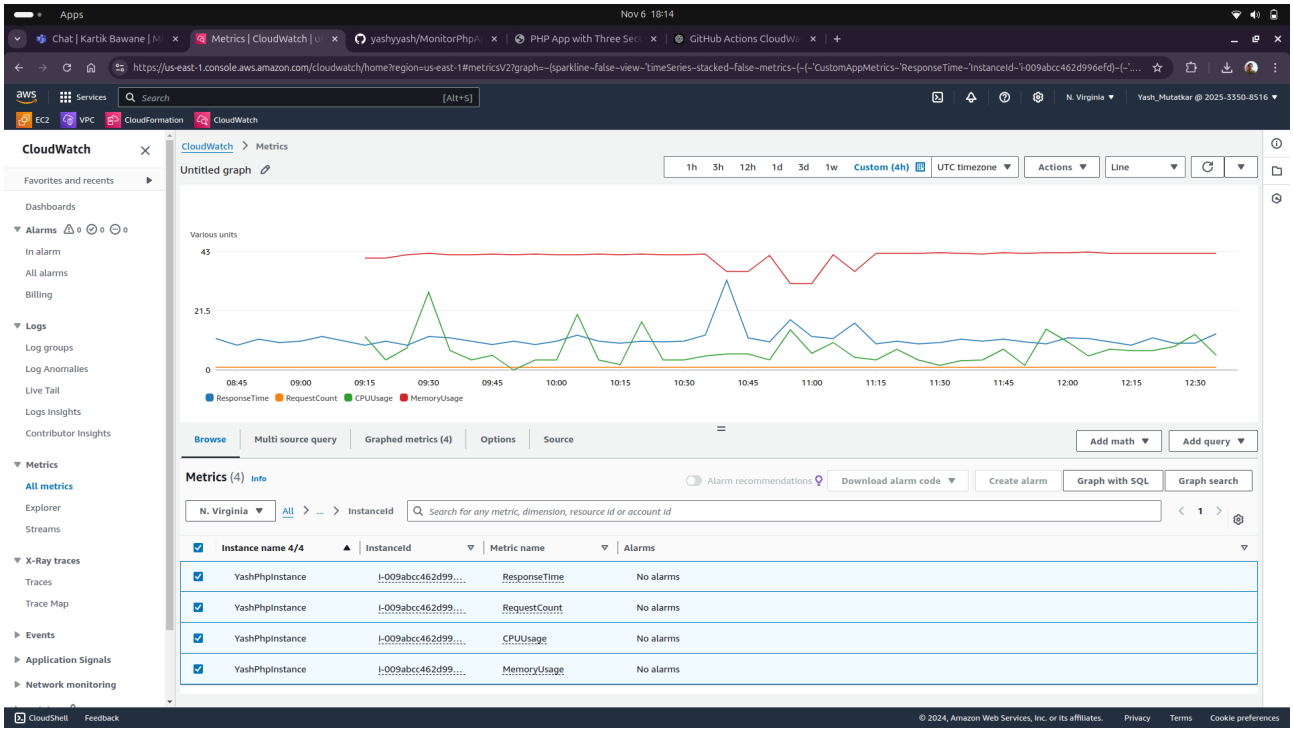
3. **AWS Credentials**: The workflow uses AWS credentials (stored securely in GitHub Secrets) to send data to CloudWatch.
4. **Health Check & Metrics**: The workflow checks the health of the PHP application by measuring the response time and status code. If everything's okay, it sends these metrics to CloudWatch.

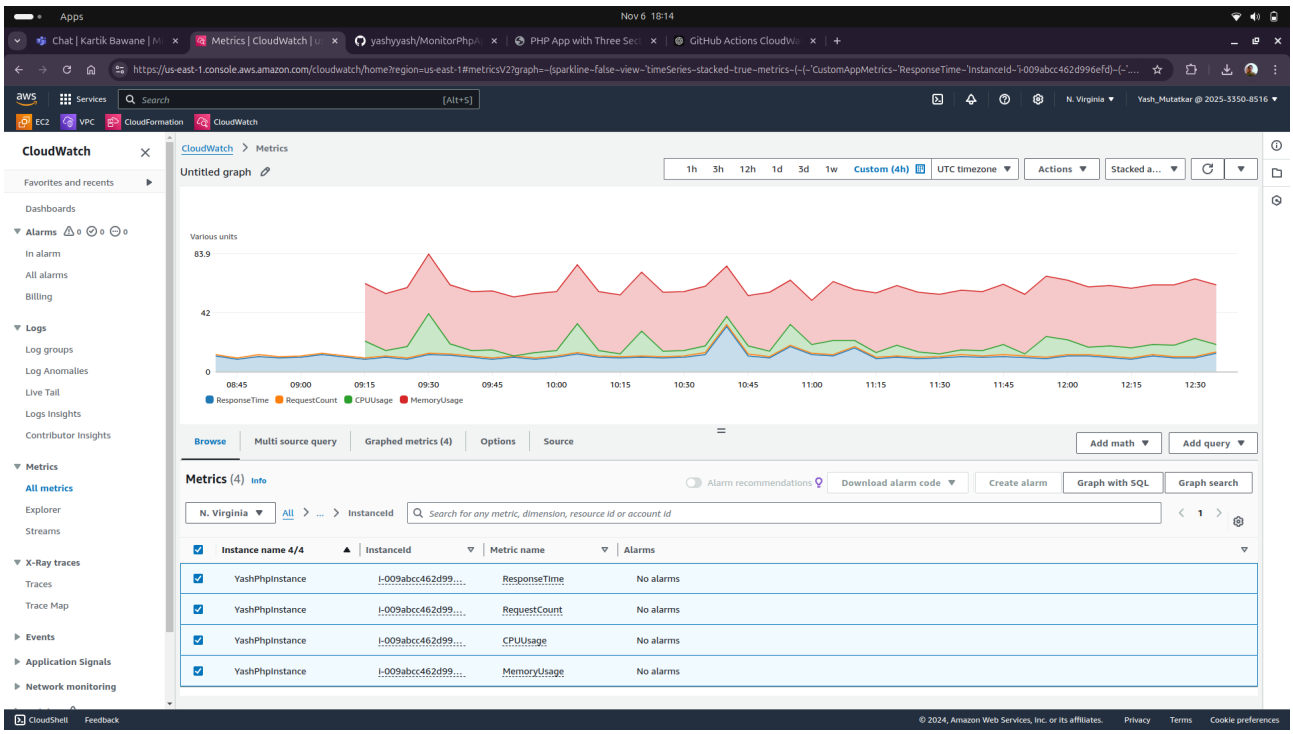


Step 6: Monitoring and Alerts

- Every 5 minutes, GitHub Actions checks the application's health and sends the metrics to CloudWatch.
- If there's a failure (like if the HTTP status code isn't 200), an **issue** is automatically created in GitHub, so I can act on it right away.
- I can also monitor the response time, system CPU, and memory usage in **CloudWatch**.


CloudWatch
Untitled graph
Actions
Data table
Label | Min | Max | Sum | Average | 11/06 12:00 | 11/06 11:55 | 11/06 11:50 | 11/06 11:45 | 11/06 11:40 | 11/06 11:35 | 11/06 11:30
ResponseTime | 9ms | 32.7ms | 501ms | 11.4ms | 11.6ms | 9.6ms | 10.2ms | 11.2ms | 10.6ms | 11.2ms | 10ms
RequestCount | 1 | 1 | 44 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1
CPUUsage | 0 | 28.3% | 240% | 7.06% | 10.3% | 14.8% | 1.82% | 7.5% | 3.64% | 3.34% | 1.82%
MemoryUsage | 31.4% | 42.8% | 1,392% | 40.9% | 42.6% | 42.8% | 42.4% | 42.7% | 42.2% | 42.4% | 42.6%
Metrics (4)
YashPhpInstance | i-009abcc462d99... | ResponseTime | No alarms
YashPhpInstance | i-009abcc462d99... | RequestCount | No alarms
YashPhpInstance | i-009abcc462d99... | CPUUsage | No alarms
YashPhpInstance | i-009abcc462d99... | MemoryUsage | No alarms


CloudWatch
Untitled graph
Actions
Line
Various units
43
21.5
0
08:45 09:00 09:15 09:30 09:45 10:00 10:15 10:30 10:45 11:00 11:15 11:30 11:45 12:00 12:15 12:30
ResponseTime RequestCount CPUUsage MemoryUsage
Metrics (4)
YashPhpInstance | i-009abcc462d99... | ResponseTime | No alarms
YashPhpInstance | i-009abcc462d99... | RequestCount | No alarms
YashPhpInstance | i-009abcc462d99... | CPUUsage | No alarms
YashPhpInstance | i-009abcc462d99... | MemoryUsage | No alarms



Outputs

By combining **AWS CloudWatch** and **GitHub Actions**, I created an automated monitoring solution for my PHP application. This setup helps me track the health and performance of my app, ensuring it runs smoothly. With CloudWatch's detailed metrics and GitHub Actions automating the process, I can receive alerts and take action whenever necessary.