Report: Amazon CloudWatch Configuration and Monitoring in AWS Environment

Name: Chaithanya M

Date: July 9, 2025

Topic: Amazon CloudWatch Configuration and Monitoring in AWS Environment

1. Introduction

Amazon CloudWatch is a comprehensive monitoring and observability service provided by AWS. It allows users to collect, visualize, and analyze metrics, logs, and events from AWS resources and on-premises infrastructure in real-time. CloudWatch plays a critical role in identifying performance bottlenecks, detecting operational issues, and automating responses to changes in resource usage.

It integrates seamlessly with services like EC2, RDS, Lambda, S3, and more, offering capabilities such as custom dashboards, alarms, anomaly detection, and automatic actions using EventBridge or Lambda functions. CloudWatch enhances system visibility, improves reliability, and supports proactive infrastructure and application management.

2. Objective

- Monitor the performance and health of AWS resources in real-time.
- Collect and analyze logs, metrics, and custom events.
- Set alarms to detect anomalies or threshold breaches and trigger automated responses.
- Improve operational insight, resource optimization, and incident response time.
- Enable centralized monitoring for multi-service AWS environments.

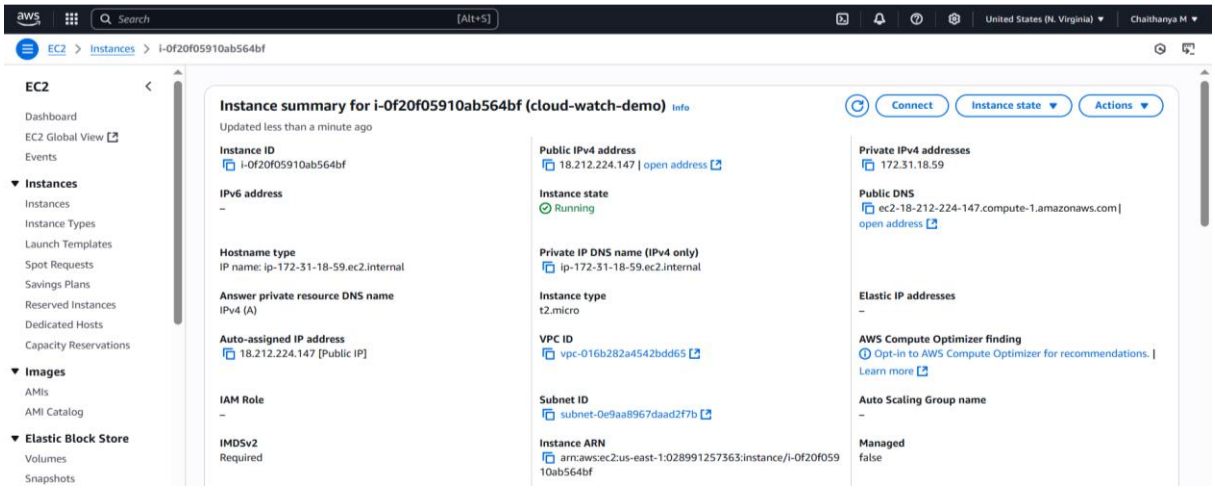
3. Prerequisites

- An active AWS account with IAM user or role having CloudWatch permissions.
- Monitored services (e.g., EC2, Lambda) must have monitoring/logging enabled.
- CloudWatch Agent installed on EC2 instances for custom metrics.
- Proper region selected, as CloudWatch metrics and logs are region-specific.

4. Implementation

Step 1: Launch an EC2 Instance

- Go to the AWS Management Console → EC2 → Launch Instance.
- Choose an Ubuntu, instance type t2.micro.
- Enable public IP.
- Add security group rule to allow SSH (port 22).
- Launch the instance using an existing or new key pair.



Step 2: Connect to the EC2 Instance

Use an SSH client or terminal: $ ssh -i newkey.pem ubuntu@18.212.224.147

Step 3: Upload and Run cpu_spike.py

- use nano in EC2 to paste: $ vi cpu_spike.py

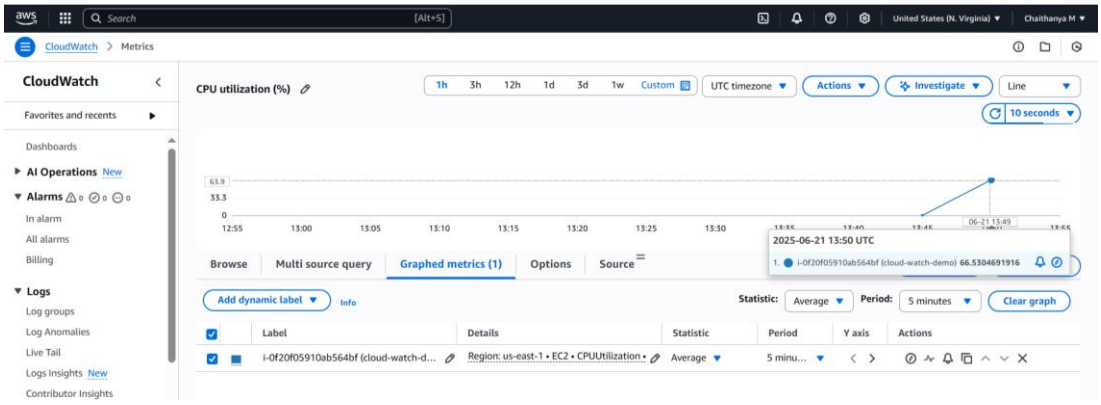
```
import time
def simulate_cpu_spike(duration=30, cpu_percent=80):
    print(f"Simulating CPU spike at {cpu_percent}%...")
    start_time = time.time()
    # Calculate the number of iterations needed to achieve the desired CPU utilization
    target_percent = cpu_percent / 100
    total_iterations = int(target_percent * 5_000_000)  # Adjust the number as needed
    # Perform simple arithmetic operations to spike CPU utilization
    for _ in range(total_iterations):
```

```
        result = 0
        for i in range(1, 1001):
            result += i
    # Wait for the rest of the time interval
    elapsed_time = time.time() - start_time
    remaining_time = max(0, duration - elapsed_time)
    time.sleep(remaining_time)

    print("CPU spike simulation completed.")

if __name__ == '__main__':
    # Simulate a CPU spike for 30 seconds with 80% CPU utilization
    simulate_cpu_spike(duration=30, cpu_percent=80)
```

- Run the script: $python3 cpu_spike.py

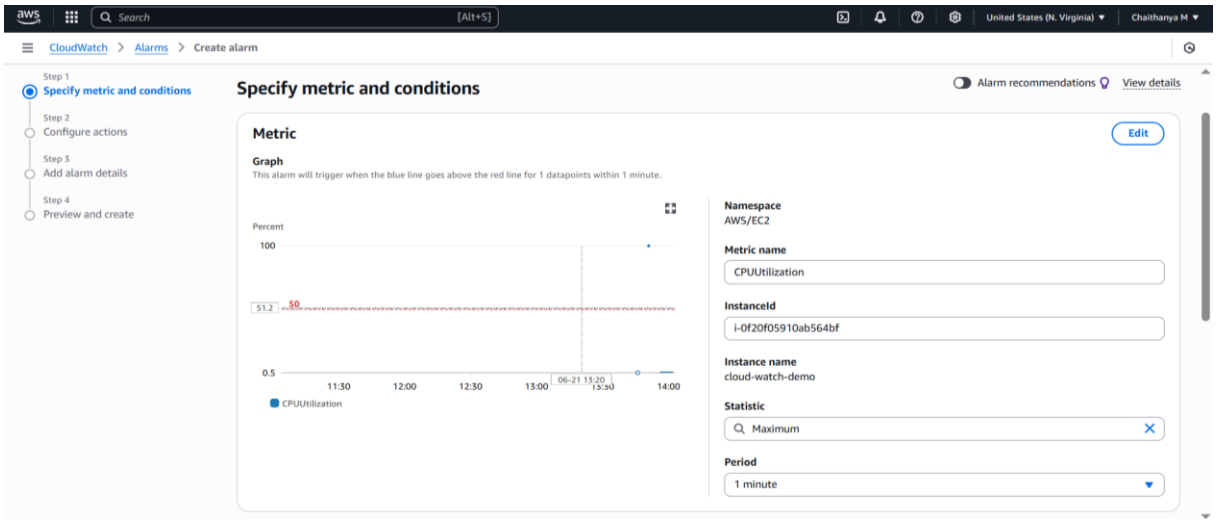
Step 4: Monitor CPU Usage in CloudWatch

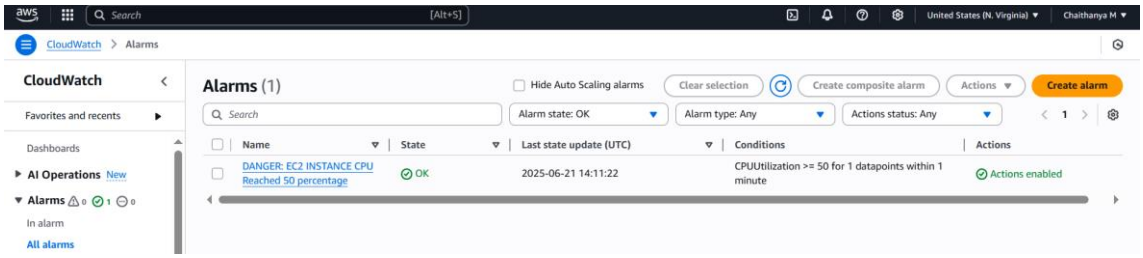
- Go to CloudWatch → Metrics → EC2 → Per-Instance Metrics**.**
- Select the metric: CPUUtilization for your EC2 instance.
- View the graph to see real-time CPU spike caused by the script.



Step 5: Create a CloudWatch Alarm

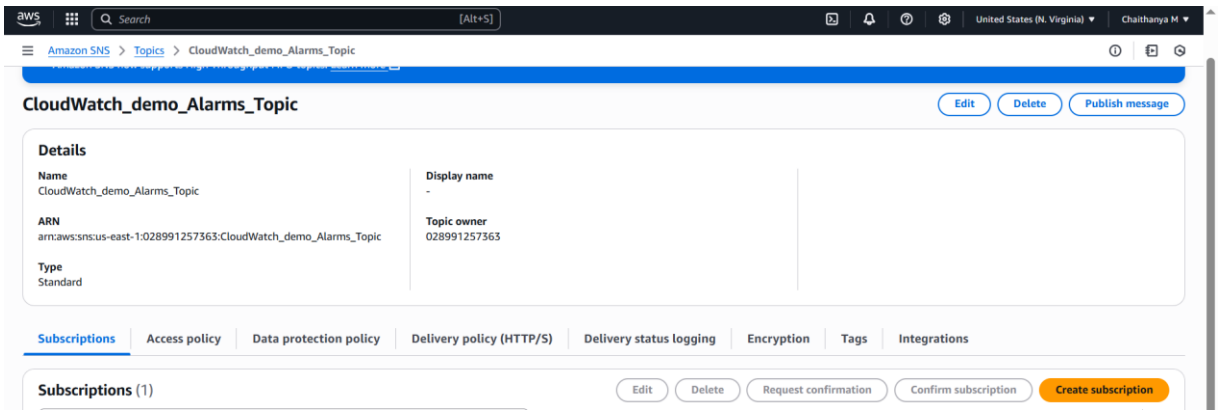
- In CloudWatch, go to Alarms → Create Alarm.
- Choose the EC2 → CPUUtilization metric.
- Set threshold (e.g., when CPU > 80% for 5 minutes).
- Choose Actions → Send a notification to an SNS topic.





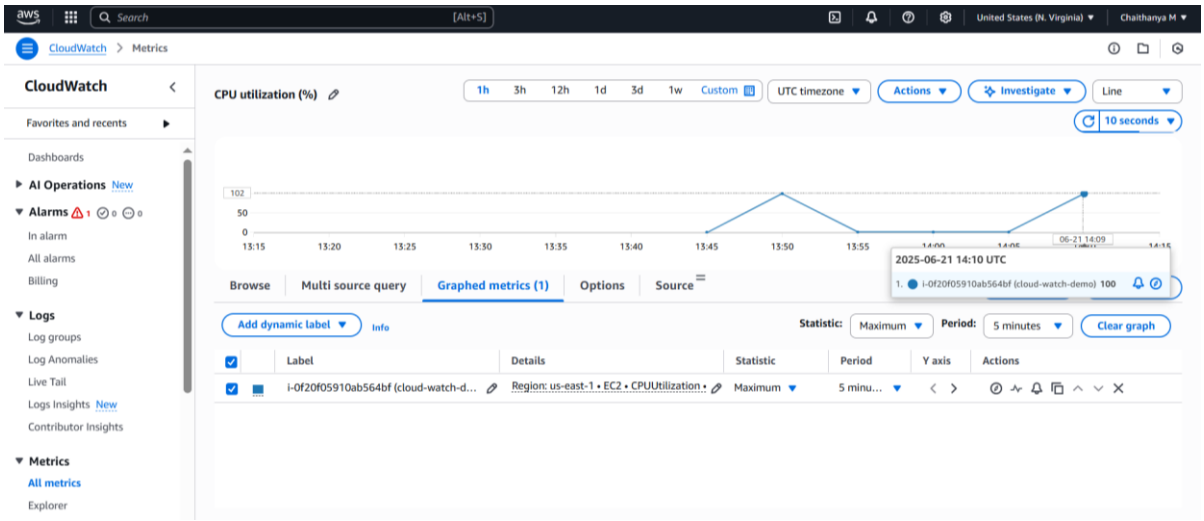
Step 6: Set Up SNS to Send Email

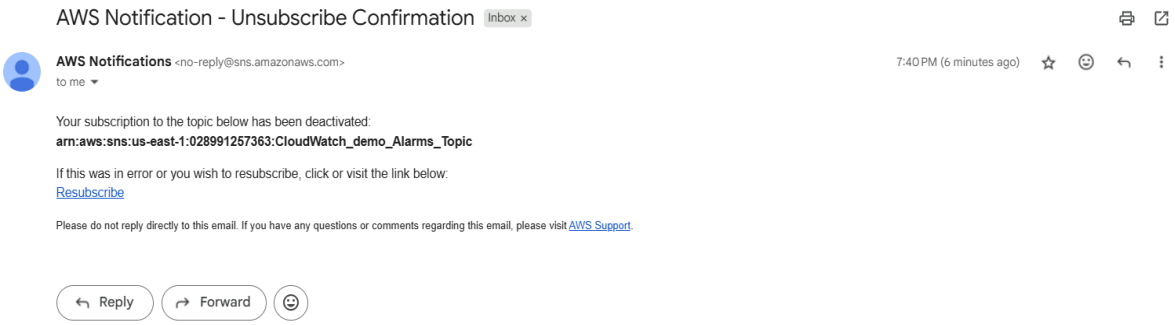
- In SNS, create a new topic.
- Add an email subscription with your email address.
- Confirm the subscription from your email inbox.

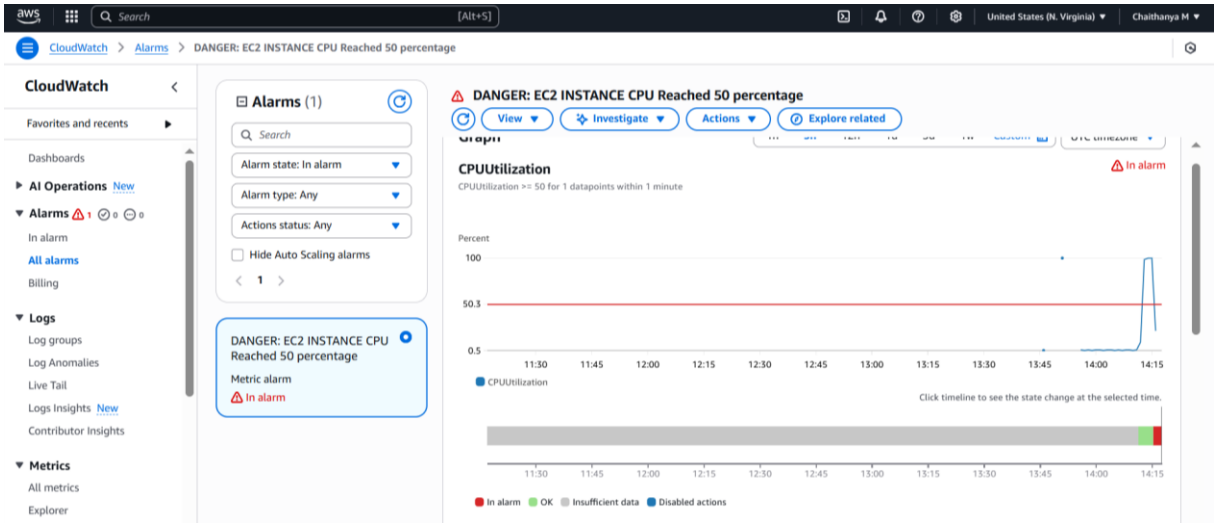


Step 7: Test the Alarm

- Keep cpuspike.py running to trigger high CPU usage.
- Once the threshold is crossed, CloudWatch will send an alert via SNS.
- You should receive an email notification.







5. Conclusion

This report outlines the practical use of Amazon CloudWatch for real-time monitoring and alerting within the AWS ecosystem. By launching an EC2 instance and simulating a high CPU load using the cpuspike.py script, we were able to observe how CloudWatch captures system metrics and enables intelligent monitoring.

The implementation showcased how to:

- View CPU utilization metrics in real time.
- Create a CloudWatch alarm based on defined thresholds.
- Integrate the alarm with Amazon SNS to send email notifications to subscribed users.

This end-to-end process demonstrates CloudWatch's capability to provide automated, responsive infrastructure monitoring without the need for manual intervention. It enables proactive system health checks, faster troubleshooting, and reduced downtime through early detection of performance anomalies.

In production environments, such configurations are essential for ensuring operational continuity and improving overall system reliability. Furthermore, extending this setup to monitor memory, disk space, application logs, and custom metrics can offer a comprehensive observability strategy for modern cloud-based applications.