**Project Summary Table**

| **Section** |  | **Details** |
| --- | --- | --- |
| **Title** |  | End-to-End AWS Monitoring & Notification System using CloudTrail, CloudWatch, SNS & EC2 Services |
| **Objective** |  | To implement centralized monitoring by:  • Tracking AWS account activity using CloudTrail  • Monitoring EC2 CPU utilization via CloudWatch Dashboard  • Monitoring Tomcat & NGINX service health via custom metrics  • Configuring CloudWatch alarms to trigger SNS email alerts for failures |
| **Prerequisites** |  | • AWS Account with admin permissions  • EC2 Instance (Amazon Linux 2023, t2.micro)  • IAM Role or Access Keys for AWS CLI  • Internet access to download Tomcat and packages  • SSH client for server access |
| **AWS Services Used** |  | • CloudTrail  • SNS (Simple Notification Service)  • EC2  • CloudWatch Metrics, Dashboard & Alarms  • S3 (for CloudTrail Logs) |
| **Tasks Performed** |  | Create CloudTrail trail and enable logging to S3 & CloudWatch Logs   Create SNS topic and confirm Email subscription   Launch EC2 instance and enable CPU metrics dashboard   Create CPU utilization alarm >70% with SNS notification & EC2 action   Install Tomcat, write custom monitoring script & push metric to CloudWatch   Configure cron job for Tomcat auto monitoring   Install NGINX & monitor service status using CloudWatch Custom Namespace   Create NGINX service alert using SNS |
| **Issues Faced** |  | AWS CLI credentials missing → Resolved using aws configure   Wrong Tomcat path in monitoring script → Updated correct version path   Custom metric showing “Insufficient data” initially → Fixed after cron job runs   HTTP service status 0 displayed until NGINX started → Verified & fixed by restarting service |
| **Security Measures** |  | • Log file validation enabled in CloudTrail  • Monitoring root account usage  • SNS alerts for critical failures |
| **Result** |  | A complete automated monitoring system built, capable of:   Detecting service failures (Tomcat/NGINX)   Detecting high CPU load   Sending real-time alerts via SNS email   Storing API logs in S3 for auditing/security |
| **Conclusion** |  | Successfully deployed a **360° AWS Monitoring System** integrating application health, EC2 performance & account security alerts using AWS native services. This setup improves operational reliability, provides early-warning notifications, and ensures continuous visibility into cloud infrastructure health. |

1.Enable CloudTrail monitoring and store the events in s3 and CloudWatch log events.

**Step 1: Open AWS CloudTrail Console**

* Navigated to **AWS CloudTrail** under **Management & Governance**
* Clicked **Create a Trail**

Purpose → To continuously log API activities in AWS account.

**Step 2: Configure Trail Name & Storage**

* Trail name provided: **monitoring-events**
* Selected **Create a new S3 bucket** to store CloudTrail logs
* Bucket automatically created with a unique name (example shown in screenshot)

Logs will be stored in S3 for long-term auditing and security analysis.

**Step 3: Enable Log File Validation**

* Checked **Log file validation**

Ensures logs are **not tampered** or **modified**.

**Step 4: Enable CloudWatch Logs Integration**

* CloudWatch Logs enabled (Optional section)
* Created a **new log group**
* IAM role auto-created for CloudTrail → CloudWatch integration

Helps in real-time monitoring and alerting using CloudWatch.

**Step 5: Select Event Types**

* Enabled **Management events**
* Enabled **Read + Write** API activity

All-important actions in AWS are now being captured.

**Step 6: Review and Create Trail**

* Trail successfully created
* Status: **Logging**

CloudTrail started recording events across all regions.

**Step 7: Verify Logs in CloudWatch Logs**

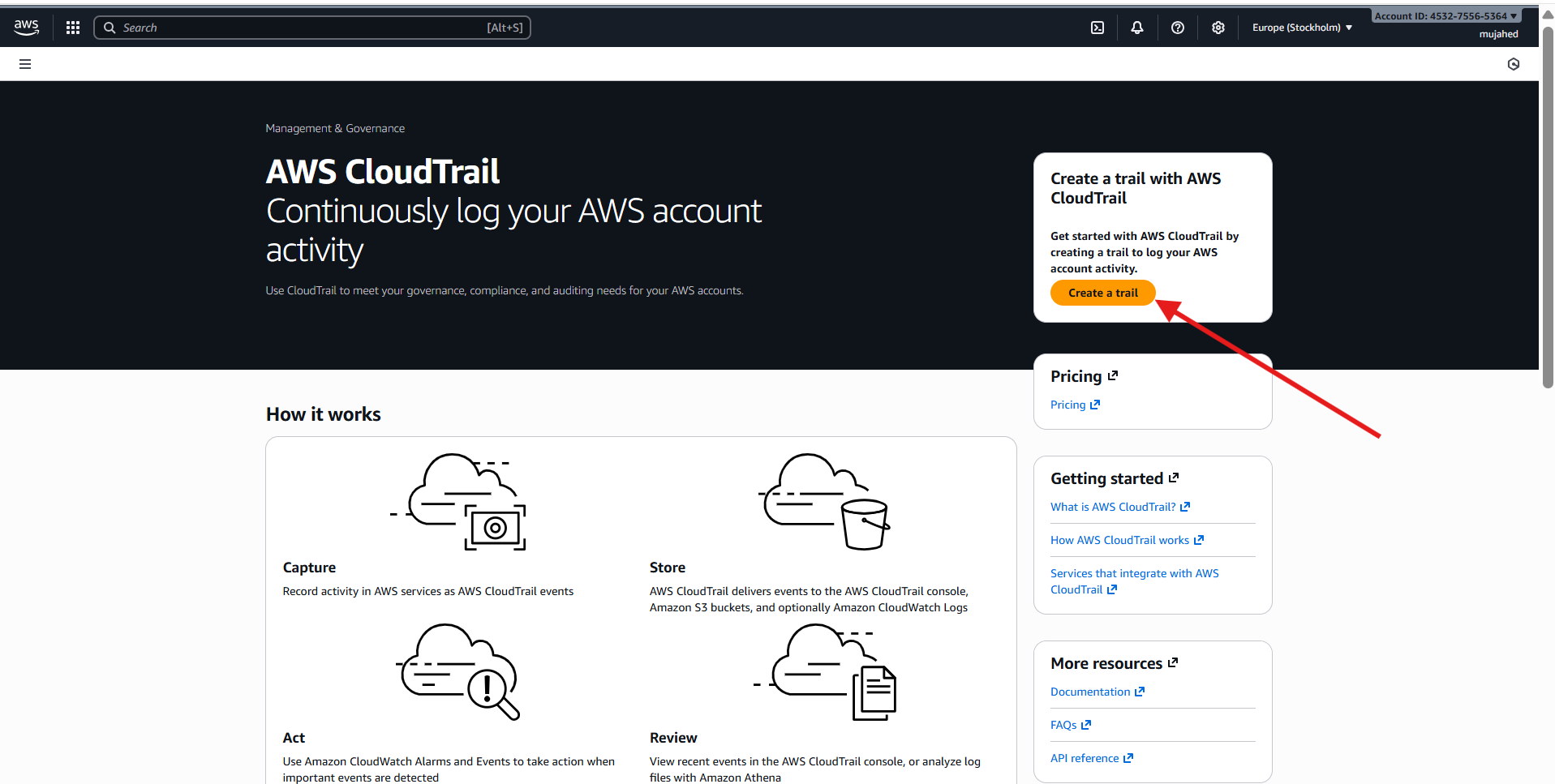
* Opened **CloudWatch → Log Groups**
* Selected the CloudTrail log group
* Event logs displayed (with API event details)

**Step 8: Verify Logs in S3 Bucket**

* Navigated to **S3 console**
* Opened auto-created CloudTrail logs bucket
* Downloaded a .json.gz file containing event data
* Viewed event details in VS Code

Contains detailed event records with:

* user Identity
* event Source
* sourceIPAddress
* request Parameters
* response Elements



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Enable SNS for cloud trial to send alert on email.

**Step 1: Open Amazon SNS Console**

* Navigated to **Amazon Simple Notification Service (SNS)** from the AWS Management Console.
* Clicked **Create Topic** option.

SNS is used for sending notifications (Email, SMS, HTTPS, Lambda triggers, etc.)

**Step 2: Create a Standard SNS Topic**

* Topic type chosen: **Standard**
* Entered Topic Name: **cloudtrail-alerts**
* Clicked **Create topic**

This topic will receive CloudTrail alerts and push notifications to subscribed endpoints.

**Step 3: Create a Subscription**

* On the Topic details page → clicked **Create subscription**
* Protocol selected: **Email**
* Provided Email address: **(your Gmail ID)**  
  Example from screenshot: *mohdmuj…@gmail.com*
* Clicked **Create subscription**

This creates a subscription request—but it must be verified via email.

**Step 4: Confirm Subscription via Email**

* Opened Gmail inbox
* Received **AWS Notification – Subscription Confirmation**
* Clicked **Confirm subscription** link

This validates the subscription so alerts can be delivered.

**Step 5: Subscription Confirmation Successful**

* AWS confirmation page shows:  
   “Subscription confirmed!”

Now this email will receive CloudTrail event notifications once alerts are configured.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a email

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Configure cloud watch monitoring and record the CPU utilization and other metrics of ec2.

**Step 1: Launch an EC2 Instance**

* Opened **EC2 → Launch instance**
* Entered name: **monitor-cpu**
* Selected AMI: **Amazon Linux 2023 (Free tier eligible)**
* Selected instance type: **t2.micro**
* Kept other default settings and clicked **Launch instance**

Your instance is now running and will produce CPU metrics.

**Step 2: View Monitoring for the Instance**

* Navigated to **Instances → monitor-cpu**
* Clicked the **Monitoring** tab

Shows basic CloudWatch metrics like CPUUtilization.

**Step 3: Open CloudWatch Console**

* Navigated to **CloudWatch**
* Clicked **Metrics → All Metrics**
* Selected service: **EC2**

CloudWatch is used to track performance metrics of AWS resources.

**Step 4: Select EC2 Per-Instance Metrics**

* Selected **Per-Instance Metrics**
* Chose the metric **CPUUtilization** of your EC2 Instance

This selects CPU percentage performance over time.

**Step 5: Add Metric to Graph / Dashboard**

* Clicked **Actions → Add to dashboard**
* Created a new dashboard name: **Monitor-metrics**
* Set **Custom (1m)** refresh time
* Applied changes

Dashboard created to visually monitor CPU performance.

**Step 6: Generate CPU Load on EC2**

* Connected to EC2 instance using SSH
* Ran the following command repeatedly:

yes > /dev/null &

(or multiple times)

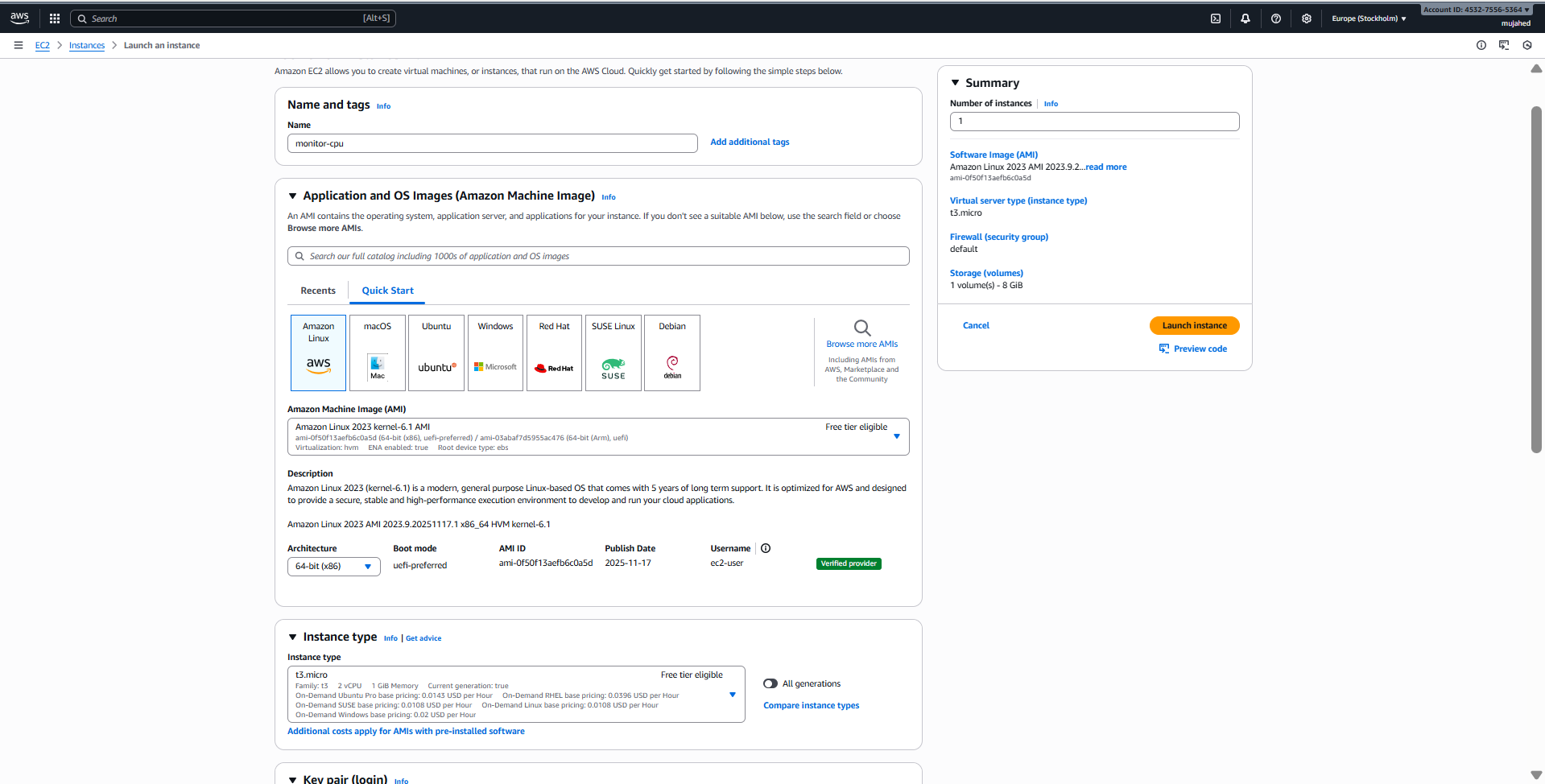
* Used top command to verify **CPU usage increased**

The yes command puts high load on CPU.

**Step 7: Check CPU Usage on Dashboard**

* Returned to CloudWatch Dashboard
* CPUUtilization graph showed increased values (e.g., **57.9%**)

CloudWatch successfully monitored real-time CPU load from EC2.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A black background with a black square

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Create one alarm to send alert to email if the CPU utilization is more than 70 percent.

**Step 1: Open CloudWatch Alarms**

* Go to **CloudWatch Console**
* Click **Alarms** (left menu)
* Click **Create alarm**

This starts the alarm creation workflow.

**Step 2: Select Metric**

* Click **Select metric**
* Choose:
  + **EC2 → Per-Instance Metrics**
  + Select **CPUUtilization** metric of your instance: *monitor-cpu*
* Click **Select metric**

The alarm will monitor CPU percentage of selected EC2 instance.

**Step 3: Define Alarm Conditions**

* Threshold type: **Static**
* Condition: **Whenever CPUUtilization is... Greater**
* Threshold value: **70**
* Statistic: **Average**
* Period: **5 minutes**

Alarm goes to **In Alarm** state if CPU > 70% for 5 minutes.

**Step 4: Configure Notification Actions**

* Alarm state trigger: **In alarm**
* Choose **Send a notification to** your existing SNS topic:
  + **cloudtrail-alerts**
* Email already subscribed → you will receive alerts

This triggers SNS message when condition is reached.

**Step 5 (Optional): Add EC2 Actions**

* Enabled EC2 Action: **Stop this instance** or **Terminate instance**
  + In your screenshot → **Terminate instance** selected

Use carefully — this will automatically shut down the instance when CPU is high!

**Step 6: Provide Alarm Name**

* Alarm name: **cpu-usage-greater-than-70%**
* Description: *(optional)*

Easy identification in console.

**Step 7: Review and Create**

* Review alarm configuration
* Click **Create alarm**

Alarm is now active and monitoring metrics.

**Step 8: Alarm Created & Active**

* Alarm state: **OK**
* Shows last update and graph under **Details**

It will move to **ALARM** state when CPU spikes above threshold

A computer screen with a red arrow pointing to a message

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Create Dashboard and monitor tomcat service whether it is running or not and send the alert.

**Step 1: Install Java on EC2**

You installed Amazon Corretto (Java) required for Tomcat:

sudo yum install java-17-amazon-corretto -y

Java installed successfully

**Step 2: Download and Install Apache Tomcat**

Downloaded Tomcat package:

wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.112/bin/apache-tomcat-9.0.112.tar.gz

Extracted and moved:

tar -xvzf apache-tomcat-9.0.112.tar.gz

mv apache-tomcat-9.0.112 /opt/

Tomcat installed in: /opt/apache-tomcat-9.0.112/

**Step 3: Start Tomcat Server**

cd /opt/apache-tomcat-9.0.112/bin/

./startup.sh

Verified Tomcat running successfully

**Step 4: Create Custom Monitoring Script**

Created **monitoring.bash** script:

vi monitoring.bash

Script Content

#!/bin/bash

# Get Instance ID

INSTANCE\_ID=$(ec2-metadata -i | cut -d " " -f2)

checkTomcatStatus(){

counter=0

ps aux | grep "/opt/apache-tomcat-9.0.112/" | grep -v grep | while read -r LINE

do

counter=$((counter+1))

done

echo $counter

}

n=$(checkTomcatStatus)

if [ "$n" -eq 0 ]

then

aws cloudwatch put-metric-data --region eu-north-1 --metric-name tomcat --value 0 --namespace tomcat --dimensions InstanceId=$INSTANCE\_ID

else

aws cloudwatch put-metric-data --region eu-north-1 --metric-name tomcat --value 1 --namespace tomcat --dimensions InstanceId=$INSTANCE\_ID

fi

Made script executable:

chmod 777 monitoring.bash

**Step 5: Configure AWS CLI**

Since CloudWatch push needed credentials — you configured using:

aws configure

Entered Access Key, Secret Key, Region, Output format

**Step 6: Test script manually**

bash monitoring.bash

Script executed successfully  
 Custom CloudWatch Metric created: **Namespace → tomcat**

**Step 7: Create a CloudWatch Dashboard**

* Go to CloudWatch → Dashboards
* Click **Create dashboard**
* Name: **monitor-tomcat**
* Add metric widget → Select namespace **tomcat**
* Select metric → InstanceId → **visualized value (0 or 1)**

Dashboard shows Tomcat status in real-time

**Step 8: Schedule Script with Cron**

Install cron:

yum install cronie -y

Start & enable cron service:

systemctl start crond

systemctl enable crond

Edit cron jobs:

crontab -e

Add job to run script every 1 minute:

\* \* \* \* \* /root/monitoring.bash

Automated monitoring configured

A black screen with white text

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a black screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Create Dashboard and monitor nginx service to send the alert if nginx is not running.

**Step 1: Install NGINX on EC2**

sudo yum install nginx -y

Installed successfully

**Step 2: Start & Enable NGINX Service**

sudo systemctl start nginx

sudo systemctl enable nginx

sudo systemctl status nginx

Verified Active status  
 NGINX running in background

**Step 3: Create Custom Monitoring Script**

Create script:

vi nginx.bash

Script content used

#!/bin/bash

INSTANCE\_ID=$(ec2-metadata -i | cut -d " " -f2)

n=$(ps aux | grep nginx | grep -v grep | wc -l)

if [ "$n" -eq 0 ]

then

aws cloudwatch put-metric-data --region eu-north-1 --metric-name nginx\_status --value 0 --namespace nginx --dimensions InstanceId=$INSTANCE\_ID

else

aws cloudwatch put-metric-data --region eu-north-1 --metric-name nginx\_status --value 1 --namespace nginx --dimensions InstanceId=$INSTANCE\_ID

fi

Make executable:

chmod 777 nginx.bash

Test script:

bash nginx.bash

Custom metric created successfully in **CloudWatch namespace: nginx**

**Step 4: Configure Cron-job to Run Script Automatically (every minute)**

Install cron:

yum install cronie -y

Enable cron:

systemctl enable crond

systemctl start crond

Schedule script:

crontab -e

Add job:

\* \* \* \* \* /root/nginx.bash

Continuous monitoring activated

**Step 5: Create CloudWatch Dashboard for NGINX Status**

* Go to **CloudWatch → Dashboards**
* Create dashboard: **monitor-nginx**
* Add widget → **Custom namespaces → nginx**
* Select metric: **nginx\_status**

Widget now shows:

* **1 → NGINX Running**
* **0 → NGINX Stopped**

**Step 6: Create CloudWatch Alarm**

* Go to **CloudWatch → Alarms → Create alarm**
* Select metric:  
  Namespace → **nginx**  
   Metric → **nginx\_status**
* Condition:
  + Whenever **nginx\_status is Lower than 1**
  + Period: 5 minutes

Alarm triggers when NGINX stops

**Step 7: Configure Notification (SNS Alert)**

* Choose alarm state: **In alarm**
* Select SNS topic: cloudtrail-alerts
* Email already subscribed → Alerts delivered to Gmail

Email alert sent instantly when service fails

A screenshot of a computer

AI-generated content may be incorrect.

A black screen with white text

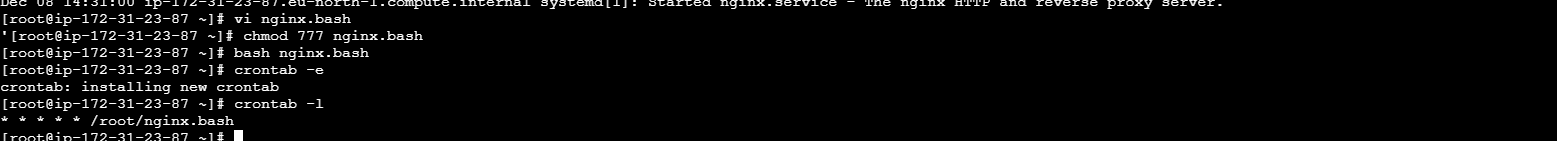
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A computer screen shot of a computer screen

AI-generated content may be incorrect.



A white background with blue and black text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.