MadMobile

Assignment

**Table of Contents**

[**Task 1: System Performance Optimization and Automation** 4](#_Toc176950489)

[**Analysis** 4](#_Toc176950490)

[**Identify Bottlenecks** 4](#_Toc176950491)

[**Monitoring Setup** 6](#_Toc176950492)

[**Implementing CloudWatch to continuously track key performance metrics (CPU usage, memory usage, network latency)** 6](#_Toc176950493)

[**Logs these metrics in a structured format (e.g., JSON, CSV).** 11](#_Toc176950494)

[**Triggers an alert (e.g., email, SMS) when certain thresholds are exceeded.** 15](#_Toc176950495)

[**Optimization** 17](#_Toc176950496)

[**Resource Scaling** 17](#_Toc176950497)

[**Optional Enhancements** 21](#_Toc176950498)

[**Log Rotation** 21](#_Toc176950499)

[**System Simulation** 23](#_Toc176950500)

[Write a script to simulate peak load conditions and measure the system’s performance, allowing you to test the effectiveness of your optimizations. 23](#_Toc176950501)

[**Automated Deployment** 25](#_Toc176950502)

[Task 2: Process Improvement and Communication 28](#_Toc176950503)

[**Requirements** 28](#_Toc176950504)

[**Process Review** 28](#_Toc176950505)

[Assumptions 28](#_Toc176950506)

[**Weakness Identification: Analyze the current incident management process, identifying key weaknesses (e.g., lack of automation, communication gaps, role ambiguities).** Based on the implemented incident management process, the following points can be highlighted as key weaknesses. 29](#_Toc176950507)

[**Improvement Proposal:** 30](#_Toc176950508)

[**Success Metrics:** **Define metrics to measure the success of the proposed improvements (e.g., reduced incident resolution time, improved response times).** 31](#_Toc176950509)

[**Automation** 32](#_Toc176950510)

[**Incident Automation Script:** 32](#_Toc176950511)

[**Auto-Assignment: A script that automatically assigns incidents to the appropriate team based on predefined criteria.** 32](#_Toc176950512)

[**Incident Escalation: A script that escalates incidents based on severity and elapsed time since the incident was logged.** 35](#_Toc176950513)

[**Integration: Describe how this script could be integrated into the existing incident management system.** 36](#_Toc176950514)

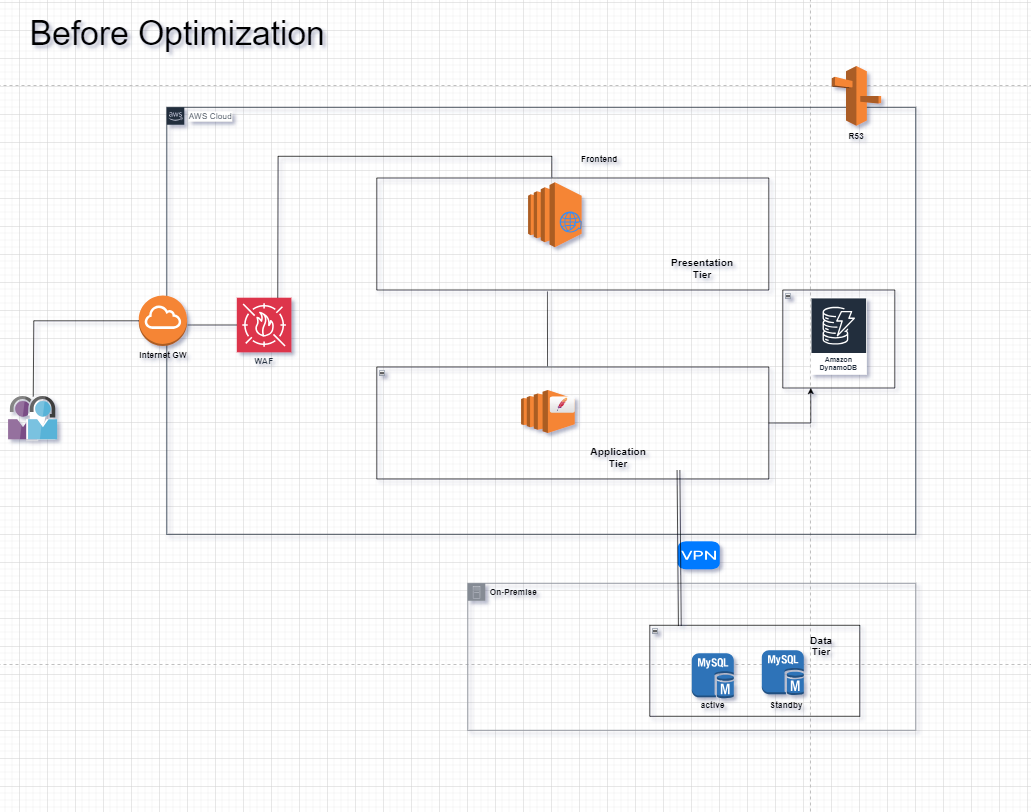
[**Optional Enhancements:** 36](#_Toc176950515)

[**Communication and Training Plan for New Incident Management Processes:** 37](#_Toc176950516)

**Task 1**

Assumptions:-

Assuming e-commerce application with three tiers:



Presentation Tier: A web server (Apache HTTP Server) running on cloud instance (AWS EC2), which does the heavy lifting of taking care of user requests and showing the website.

Application Tier: A cloud instance (AWS EC2) running a Java-based application server (Apache Tomcat), which processing business logic and communicating with database.

Data Tier: A cloud-based relational database (AWS RDS) running MySQL, responsible for storing and retrieving data.

# **Task 1: System Performance Optimization and Automation**

# **Analysis**

## **Identify Bottlenecks**

**Analyze the potential bottlenecks in a typical multi-tier application (e.g., CPU, memory, network latency, database contention) and explain how they could affect performance.**   
  
A multi-tier application can become a performance bottleneck at any layer, due to the CPU memory, network latency. disco I/O or contention issues.  
Potential bottlenecks can occur in any tier (Presentation, Application or Data) within a multi-tier application architecture and it would affect the overall system performance particularly during peak load timings.

**CPU Bottlenecks**

Handling a large number of concurrent user requests, performing resource-intensive tasks, such as image processing or video transcoding, inefficient server-side rendering or caching mechanisms or serving dynamic content may cause this issue.

**Impact on Performance:** Higher usage of CPU in web server increases the time for page render, time out and therefore a bad impression created in the client’s side. It is a condition where the server cannot accommodate all the user coming to it, and in most cases lead to slow responses, low throughput, and high latency. This is especially important in e-commerce that is the quality of the user experience has a direct impact on revenues.

The application server also makes the processing of business logic slower while also introducing the possibility of timeouts on the request. This reduces the overall response of the application and if the CPU is overloaded continuously it leads to server crash which is not good for the application availability and reliability.

Database tier can result in slothfulness of queries, delay in the amount of time it takes to fetch data, and hence have a toll on the general functionality of the application. In severe cases, this can lead to database server crashes, affecting the availability of data and causing downtime.  
  
If there are CPUs in any tier, they create a bottleneck in the multi-tier application and response times are slow, timeouts occur or the application stops working.” Determination of which process consumes a lot of CPU is crucial because it can be because of the numerous, simultaneous requests processed, or due to the execution of some complex computation, or poor optimization.

**Memory Bottlenecks**

Inefficient caching or session management, Leaks or excessive object creation, Inadequate heap size or inefficient memory allocation can lead to Memory bottlenecks.  
Eg:-  
Lack of sufficient amount of memory in the web server could lead to response time issue or even the server freezing. Still when memory is exhausted the server may swap, which seriously affects its performance as we saw above. High response time or even failure is caused by having too many concurrent connections than what the Apache server can handle without adequate memory. Also if the database server is unable to have enough memory in order to cache the most accessed data, or the working set it possibly leads to occurrences of frequent disk reads that leads to latency. Low buffer size and cache may cause heavy swapping and this will affect the query performance.

**Impact on Performance:**

Insufficient memory can lead to

Frequent page reloads or slow response times.  
Inadequate caching or session management.  
Frequent disk reads, causing latency.  
Due to the excessive swapping, impacting query performance.

**Network Latency Bottlenecks**

Distance between cloud instances or data centers / geographical distance between users and the server, high traffic, Insufficient network bandwidth (bandwidth limitations) or packet loss, slow third-party APIs or services, High latency in database queries or storage access can lead to Network Latency Bottlenecks.

**Impact on Performance:**

Between the web server and clients could result in slow page load times

between the application server and other tiers (e.g., database or web server) could cause delays in processing requests.

Between the application server and the database can cause delays in query execution and data retrieval.

**Database Contention Bottlenecks**

High concurrency or locking issues, Slow queries or inadequate query optimization (Complex or unoptimized SQL queries), Inefficient database schema or indexing (leading to full table scans) , Insufficient database connection pooling (High transaction rates) or caching can leads to a Database contention bottlenecks.

***Impact on Performance:***

Slow query performance  
increased latency in data retrieval and overall degradation of the application's responsiveness.  
Lead to database server crashes, affecting the availability of data and causing downtime

**Storage Bottlenecks**

Insufficient storage capacity or high disk usage, Inefficient data retrieval or caching, Slow disk I/O or inadequate storage configuration can leads to a Storage bottlenecks.

**Impact on Performance:**  
- Insufficient storage capacity or high disk usage can cause slow query response times, leading to delayed page loads and frustrated users.  
- Slow disk I/O can increase the time it takes to retrieve data, resulting in slower application performance.  
- Due to the additional latency in the system, affecting the overall responsiveness of the application. High latency can lead to slower page loads, increased bounce rates, and a poor user experience.  
- reduce the throughput of the application, limiting the number of requests that can be processed simultaneously, **it** can result in slower performance, even during non-peak usage times.  
- increased error rates, such as timeouts or failed queries, which can cause application errors and affect user experience  
- resource contention, where multiple components of the application compete for limited storage resources. Components may need to wait for resources to become available, leading to slower response times.

## **Monitoring Setup**

### **Implementing CloudWatch to continuously track key performance metrics (CPU usage, memory usage, network latency)**

Setup custom CloudWatch custom metrics in AWS EC2

Steps: -

1. Install the CloudWatch Agent in EC2 instance.

*sudo yum install amazon-cloudwatch-agent*

2. Configure the CloudWatch Agent.

Create a configuration file manually at /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json  
amazon-cloudwatch-agent.json (<https://github.com/harshat85/MadMobileAssignment/blob/main/amazon-cloudwatch-agent.json>)

{

"metrics": {

"namespace": "CustomNamespace",

"metrics\_collected": {

"cpu": {

"measurement": [

"cpu\_usage\_idle",

"cpu\_usage\_iowait",

"cpu\_usage\_user",

"cpu\_usage\_system"

],

"metrics\_collection\_interval": 60,

"totalcpu": true

},

"mem": {

"measurement": [

"mem\_used\_percent",

"mem\_available\_percent"

],

"metrics\_collection\_interval": 60

},

"netstat": {

"measurement": [

"tcp\_established",

"tcp\_time\_wait",

"tcp\_close\_wait"

],

"metrics\_collection\_interval": 60

}

}

}

}

Novo energy

Explanation of the Configuration:  
  
Collects CPU usage metrics :   
cpu\_usage\_idle: Percentage of time the CPU is idle.

cpu\_usage\_iowait: Percentage of time the CPU is waiting for I/O operations to complete.

cpu\_usage\_user: Percentage of CPU time spent on user-level processes.

cpu\_usage\_system: Percentage of CPU time spent on system-level processes.

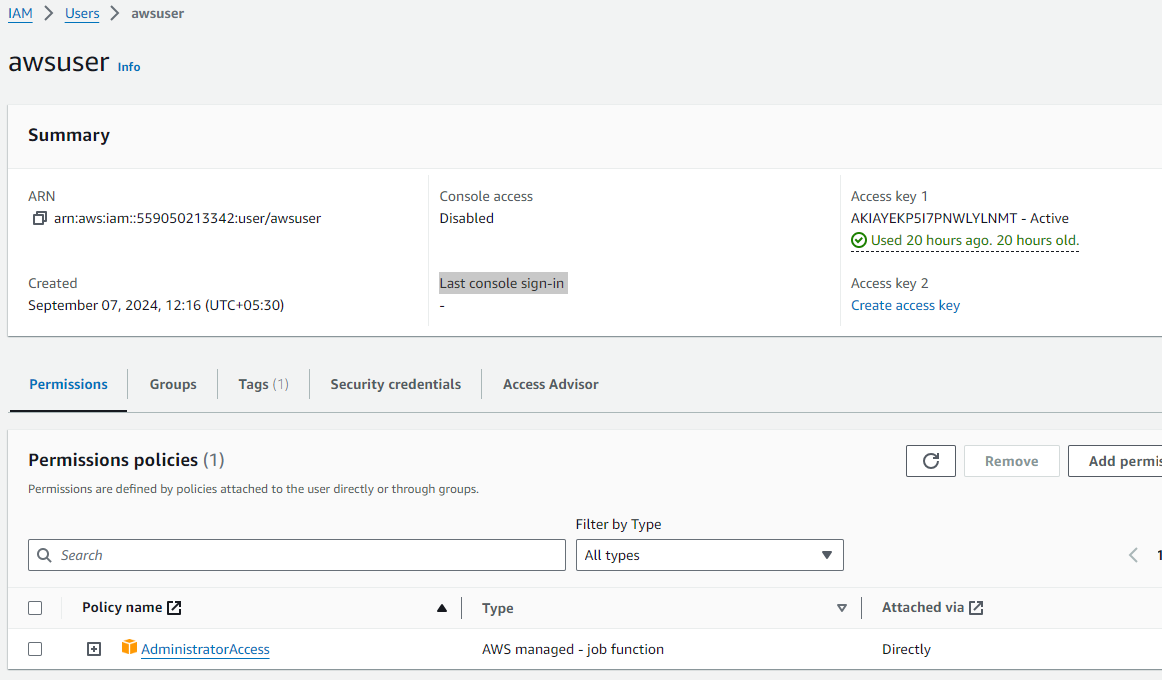
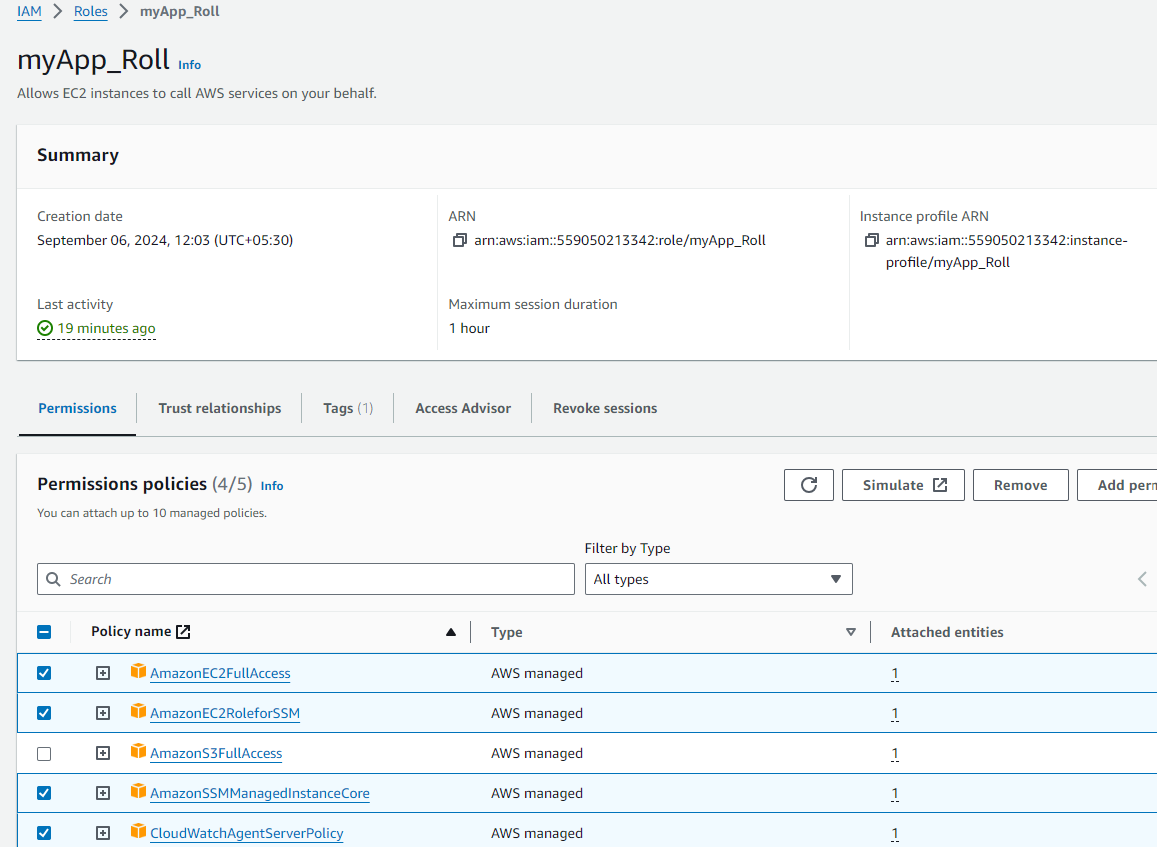
Collects memory usage metrics :  
mem\_used\_percent: Percentage of used memory.

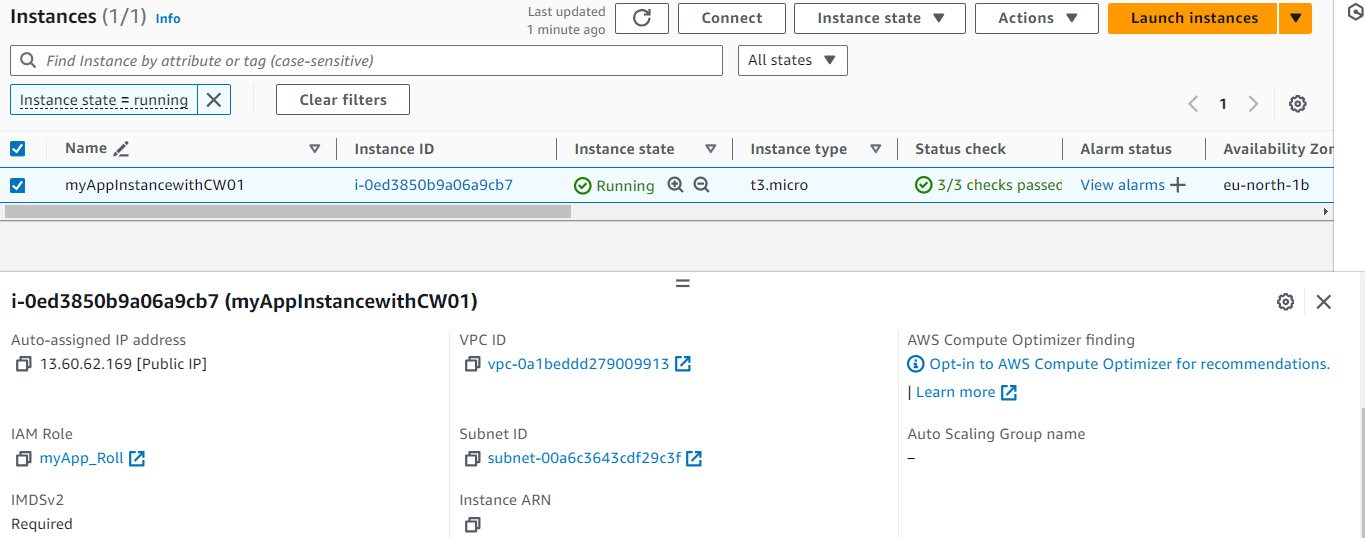
mem\_available\_percent: Percentage of available memory.  
  
Collects network-related TCP metrics :  
tcp\_established: Number of TCP connections that are currently established.

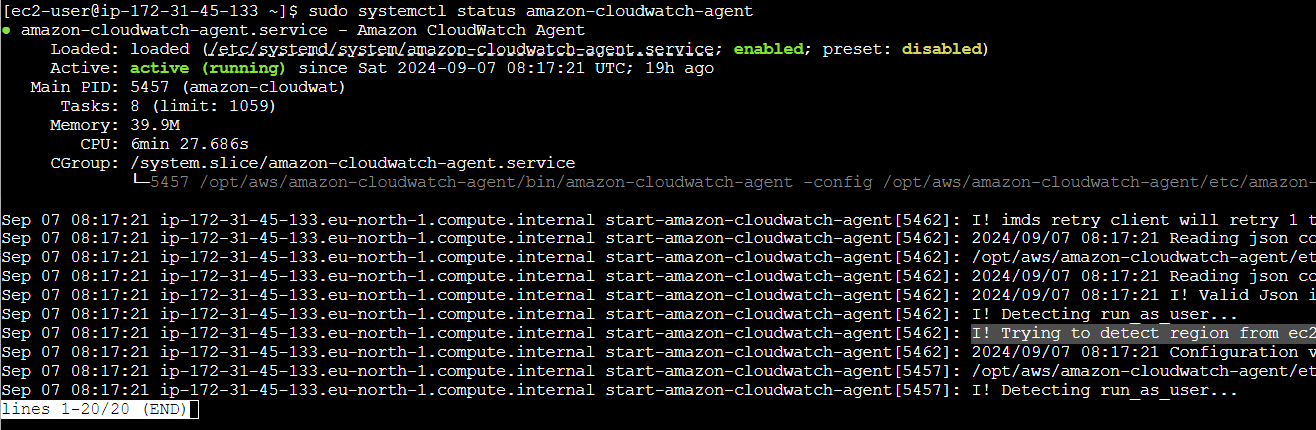
tcp\_time\_wait: Number of TCP connections that are in the TIME-WAIT state.

tcp\_close\_wait: Number of TCP connections that are in the CLOSE-WAIT state.

3. In this case I have created IAM, user (awsuser) and Role with appropriate policies (attached to the instance).  
  
AmazonEC2FullAccess  
AmazonEC2RoleforSSM  
AmazonSSMManagedInstanceCore  
CloudWatchAgentServerPolicy



4. Restart the CloudWatch Agent to apply/reload the new configuration:  
  
# sudo systemctl status amazon-cloudwatch-agent  
# sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -c file:/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json -s  
  


**5. Verify the Metrics**

- Navigate to the CloudWatch console in the AWS Management Console.  
- Check the custom namespace (CustomNamespace) to verify that the metrics for CPU usage, memory **usage, and network latency are being collected.**

**Note:   
This configuration will ensure that only the desired metrics are collected and displayed in Amazon CloudWatch under custom namespace.**

### **Logs these metrics in a structured format (e.g., JSON, CSV).**

Update the “amazon-cloudwatch-agent\_v1.json” file in /opt/aws/amazon-cloudwatch-agent/etc/ with the following content (<https://github.com/harshat85/MadMobileAssignment/blob/main/amazon-cloudwatch-agent_v1.json>)

{

"metrics": {

"namespace": "CustomNamespace",

"metrics\_collected": {

"cpu": {

"measurement": [

"cpu\_usage\_idle",

"cpu\_usage\_iowait",

"cpu\_usage\_user",

"cpu\_usage\_system"

],

"metrics\_collection\_interval": 60,

"totalcpu": true

},

"mem": {

"measurement": [

"mem\_used\_percent",

"mem\_available\_percent"

],

"metrics\_collection\_interval": 60

},

"netstat": {

"measurement": [

"tcp\_established",

"tcp\_time\_wait",

"tcp\_close\_wait"

],

"metrics\_collection\_interval": 60

}

}

},

"logs": {

"logs\_collected": {

"files": {

"collect\_list": [

{

"file\_path": "/var/log/cpu\_metrics.log",

"log\_group\_name": "CustomNamespace/CPU",

"log\_stream\_name": "{instance\_id}",

"timestamp\_format": "%Y-%m-%d %H:%M:%S",

"timezone": "UTC"

},

{

"file\_path": "/var/log/memory\_metrics.log",

"log\_group\_name": "CustomNamespace/Memory",

"log\_stream\_name": "{instance\_id}",

"timestamp\_format": "%Y-%m-%d %H:%M:%S",

"timezone": "UTC"

},

{

"file\_path": "/var/log/network\_metrics.log",

"log\_group\_name": "CustomNamespace/Network",

"log\_stream\_name": "{instance\_id}",

"timestamp\_format": "%Y-%m-%d %H:%M:%S",

"timezone": "UTC"

}

]

}

},

"force\_flush\_interval": 15

}

}

Explanation of the Configuration:

Logs Collection:

* collect\_list includes three log files:  
    
  CPU Metrics Logs: Collected from /var/log/cpu\_metrics.log.

Memory Metrics Logs: Collected from /var/log/memory\_metrics.log.

Network Metrics Logs: Collected from /var/log/network\_metrics.log.

- Each log entry will be sent to a CloudWatch Logs group under the corresponding log group names (CustomNamespace/CPU, CustomNamespace/Memory, CustomNamespace/Network).

* Logs will be timestamped using the format %Y-%m-%d %H:%M:%S in UTC

1. Install Required Tools

sudo yum install -y sysstat net-tools

sudo yum install cronie  
sudo systemctl start crond

sudo systemctl enable crond

sysstat :- mpstat, vmstat, iostat, etc., for CPU and memory usage.  
net-tools :-For network-related metrics.  
cronie :- Install cron package to setup cronjobs

1. Create log files where the metrics will be logged:

sudo touch /var/log/cpu\_metrics.log

sudo touch /var/log/memory\_metrics.log

sudo touch /var/log/network\_metrics.log

# Set permissions so that they are writable by the 'cwagent' user or the user running the cron jobs

sudo chown cwagent:cwagent /var/log/cpu\_metrics.log

sudo chown cwagent:cwagent /var/log/memory\_metrics.log

sudo chown cwagent:cwagent /var/log/network\_metrics.log

sudo chmod 664 /var/log/cpu\_metrics.log /var/log/memory\_metrics.log /var/log/network\_metrics.log

1. Create three separate scripts for collecting CPU, memory, and network metrics. Each script will log the data in JSON format.

/opt/metric-scripts/cpu\_metrics.sh

#!/bin/bash

# Collect metrics

TIMESTAMP=$(date +'%Y-%m-%d %H:%M:%S')

CPU\_USAGE\_USER=$(mpstat 1 1 | awk '/Average:/ {print 100 - $12}')

CPU\_USAGE\_SYSTEM=$(mpstat 1 1 | awk '/Average:/ {print $4}')

CPU\_USAGE\_IDLE=$(mpstat 1 1 | awk '/Average:/ {print $12}')

CPU\_USAGE\_IOWAIT=$(mpstat 1 1 | awk '/Average:/ {print $6}')

# Format and log metrics in JSON

echo "{\"timestamp\": \"$TIMESTAMP\", \"cpu\_usage\_user\": $CPU\_USAGE\_USER, \"cpu\_usage\_system\": $CPU\_USAGE\_SYSTEM, \"cpu\_usage\_idle\": $CPU\_USAGE\_IDLE, \"cpu\_usage\_iowait\": $CPU\_USAGE\_IOWAIT}" >> /var/log/cpu\_metrics.log

/opt/metric-scripts/memory\_metrics.sh

#!/bin/bash

# Collect metrics

TIMESTAMP=$(date +'%Y-%m-%d %H:%M:%S')

MEM\_USED\_PERCENT=$(free | awk '/Mem:/ {printf "%.2f", $3/$2 \* 100.0}')

MEM\_AVAILABLE\_PERCENT=$(free | awk '/Mem:/ {printf "%.2f", $7/$2 \* 100.0}')

# Format and log metrics in JSON

echo "{\"timestamp\": \"$TIMESTAMP\", \"mem\_used\_percent\": $MEM\_USED\_PERCENT, \"mem\_available\_percent\": $MEM\_AVAILABLE\_PERCENT}" >> /var/log/memory\_metrics.log

/opt/metric-scripts/network\_metrics.sh

#!/bin/bash

# Collect metrics

TIMESTAMP=$(date +'%Y-%m-%d %H:%M:%S')

TCP\_ESTABLISHED=$(netstat -an | grep ESTABLISHED | wc -l)

TCP\_TIME\_WAIT=$(netstat -an | grep TIME\_WAIT | wc -l)

TCP\_CLOSE\_WAIT=$(netstat -an | grep CLOSE\_WAIT | wc -l)

# Format and log metrics in JSON

echo "{\"timestamp\": \"$TIMESTAMP\", \"tcp\_established\": $TCP\_ESTABLISHED, \"tcp\_time\_wait\": $TCP\_TIME\_WAIT, \"tcp\_close\_wait\": $TCP\_CLOSE\_WAIT}" >> /var/log/network\_metrics.log

Make the script executable:  
  
chmod +x /opt/metric-scripts/cpu\_metrics.sh  
chmod +x /opt/metric-scripts/memory\_metrics.sh  
chmod +x /opt/metric-scripts/network\_metrics.sh

1. To run a script every 5 minutes, schedule cron (with more detailed debugging information)  
     
   sudo crontab –e

\*/5 \* \* \* \* /opt/metric-scripts/cpu\_metrics.sh >> /var/log/cron\_debug.log 2>&1

\*/5 \* \* \* \* /opt/metric-scripts/memory\_metrics.sh >> /var/log/cron\_debug.log 2>&1

\*/5 \* \* \* \* /opt/metric-scripts/network\_metrics.sh >> /var/log/cron\_debug.log 2>&1

restart cron

# sudo systemctl restart crond

1. Verify and Test Scripts

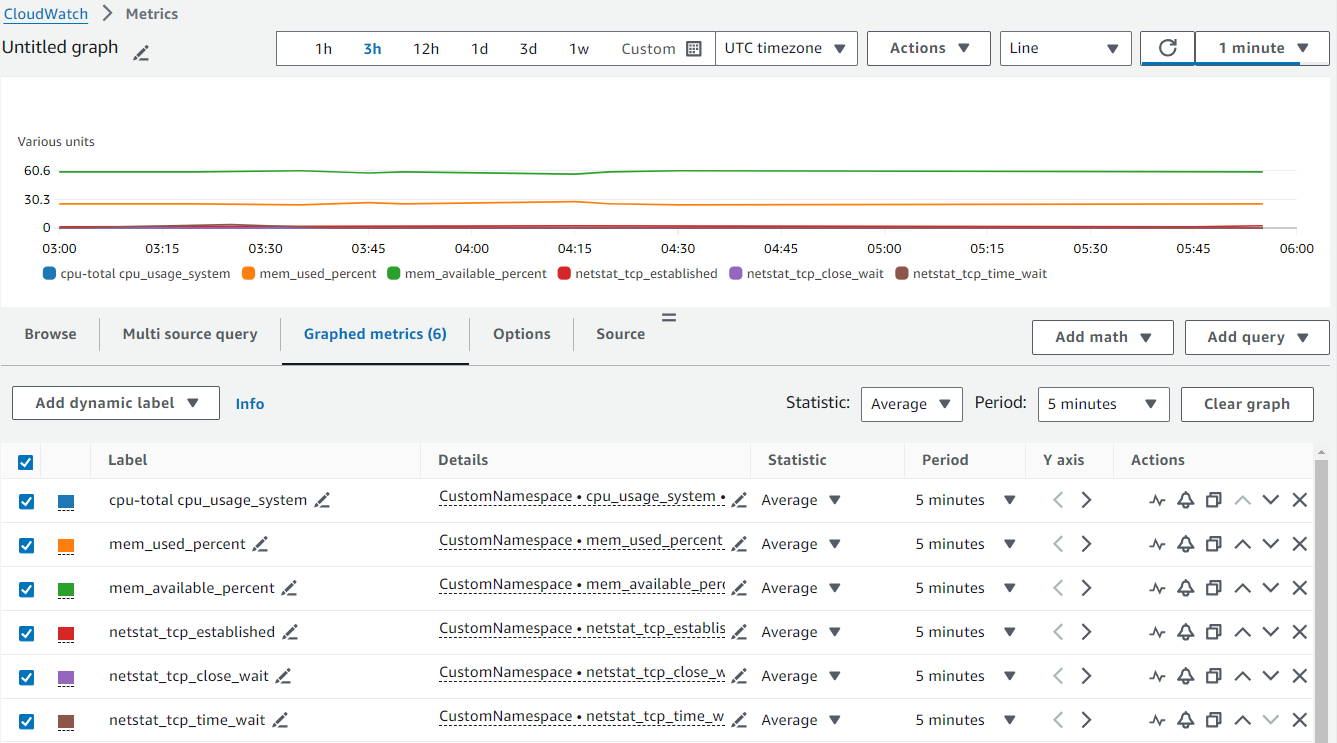
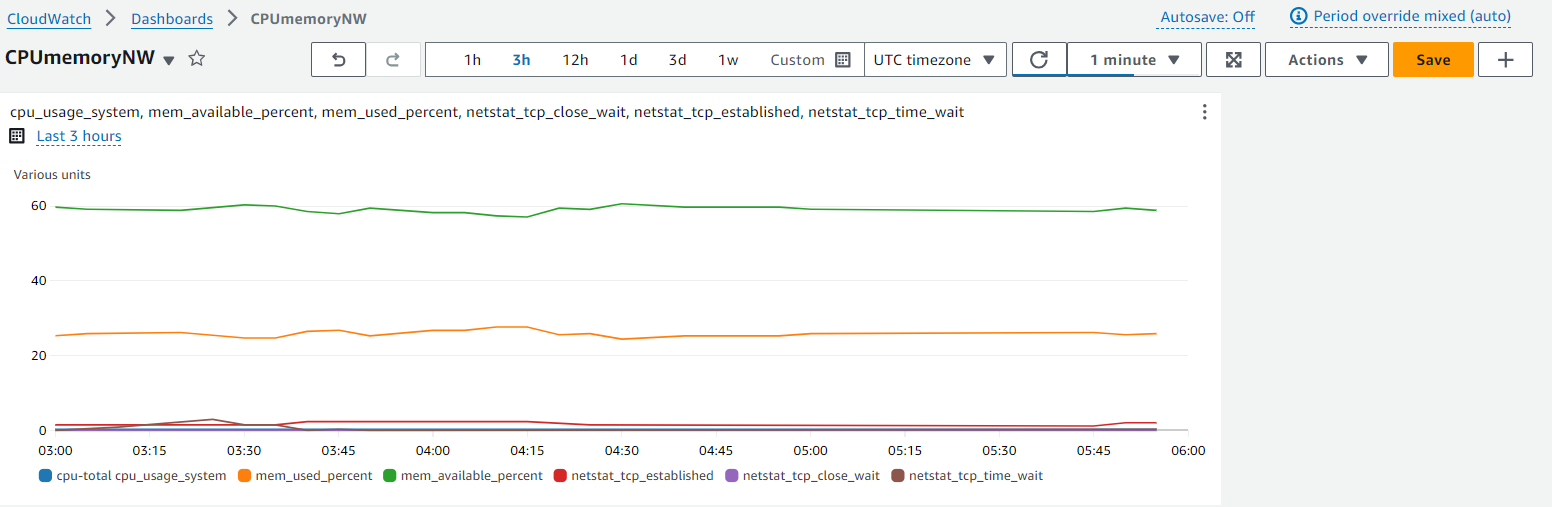
sudo bash /opt/metric-scripts/cpu\_metrics.sh

sudo bash /opt/metric-scripts/memory\_metrics.sh

sudo bash /opt/metric-scripts/network\_metrics.sh  
  
cat /var/log/cpu\_metrics.log

cat /var/log/memory\_metrics.log

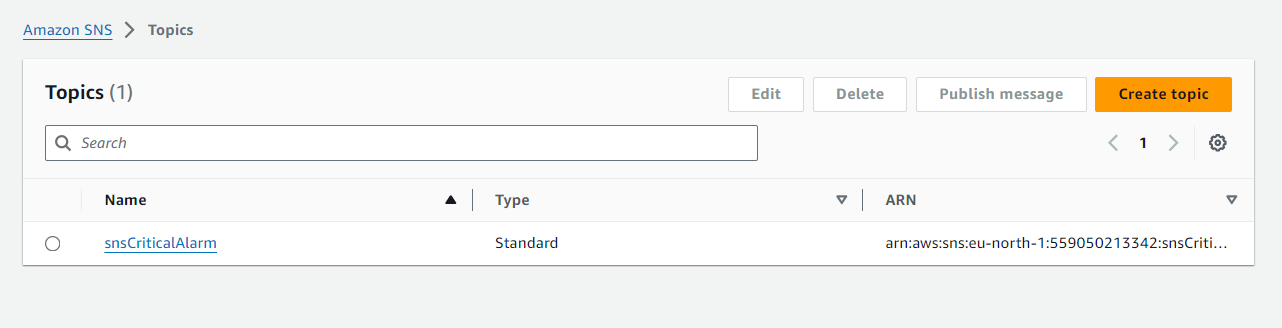
cat /var/log/network\_metrics.log

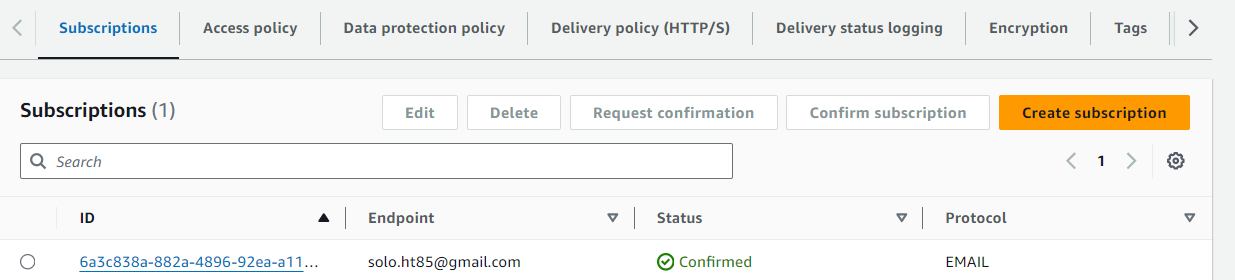
Since the CloudWatch Agent has already been configured to collect these logs and send them to CloudWatch Logs, the logs should appear in the CloudWatch console under the respective log groups (CustomNamespace/CPU, CustomNamespace/Memory, CustomNamespace/Network).  
  
  
  
  


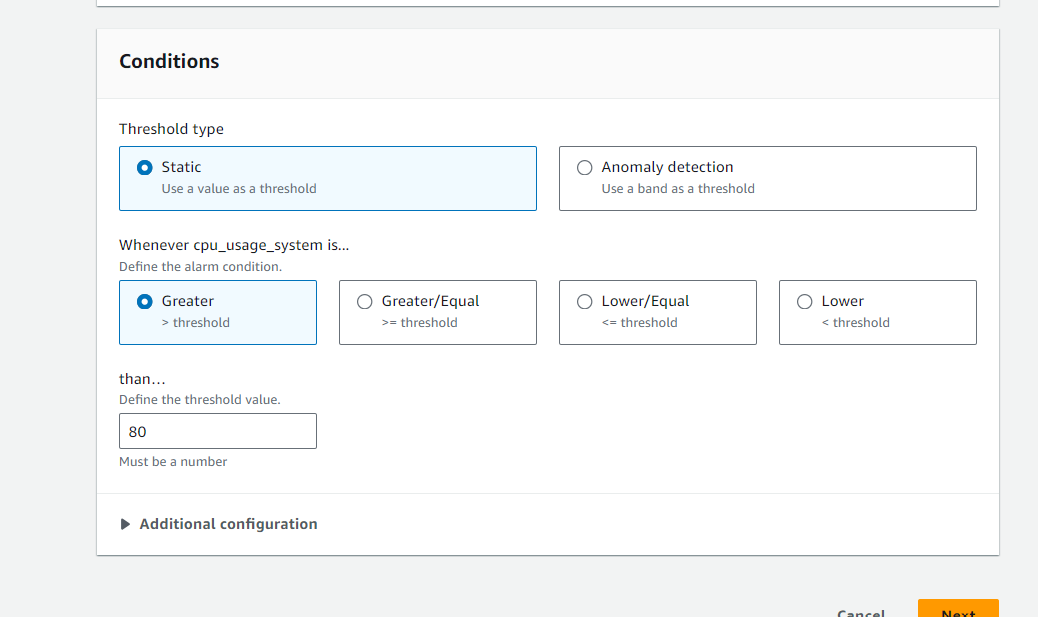
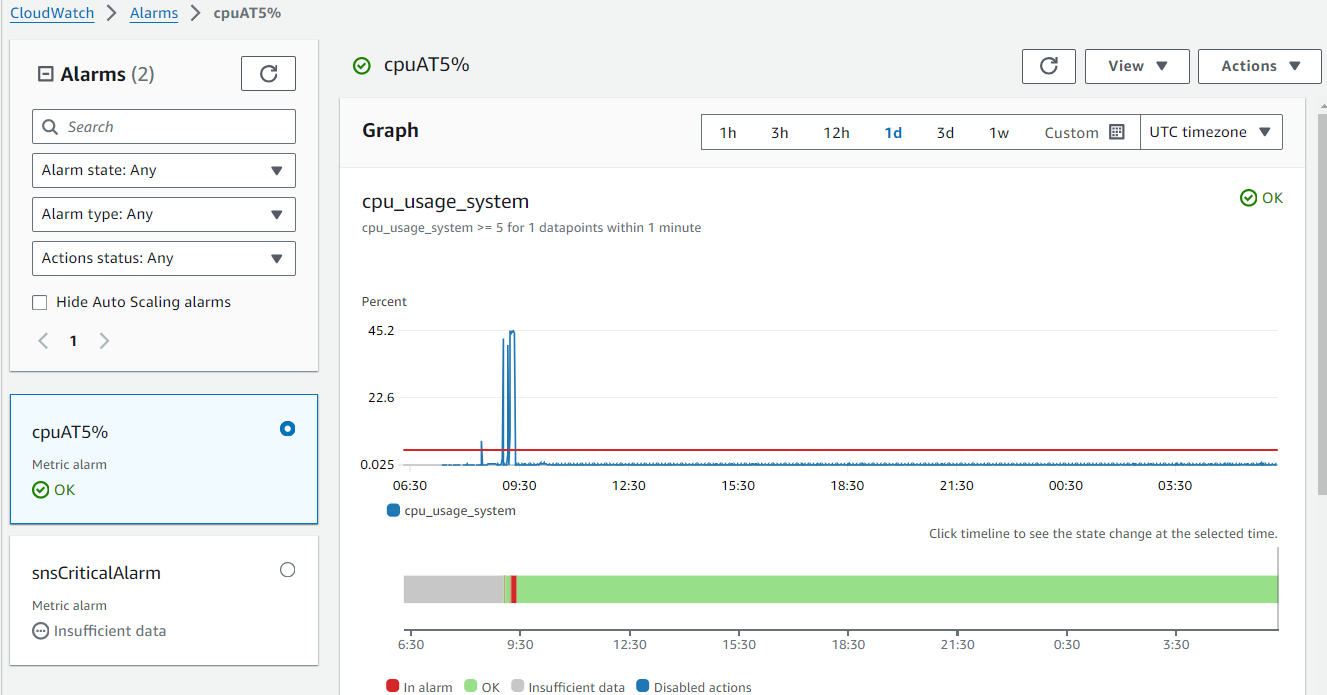
### **Triggers an alert (e.g., email, SMS) when certain thresholds are exceeded.**

1. Create a New Topic Name call (eg :- snsCriticalAlarm)

Create a Subscription:  
 Choose the Protocol (e.g., Email, SMS, HTTPS)  
 Enter the **Endpoint** (e.g., your email address or phone number)  
 Click "Create subscription"  
 Confirm the subscription if needed (e.g., by clicking a confirmation link sent to your email).



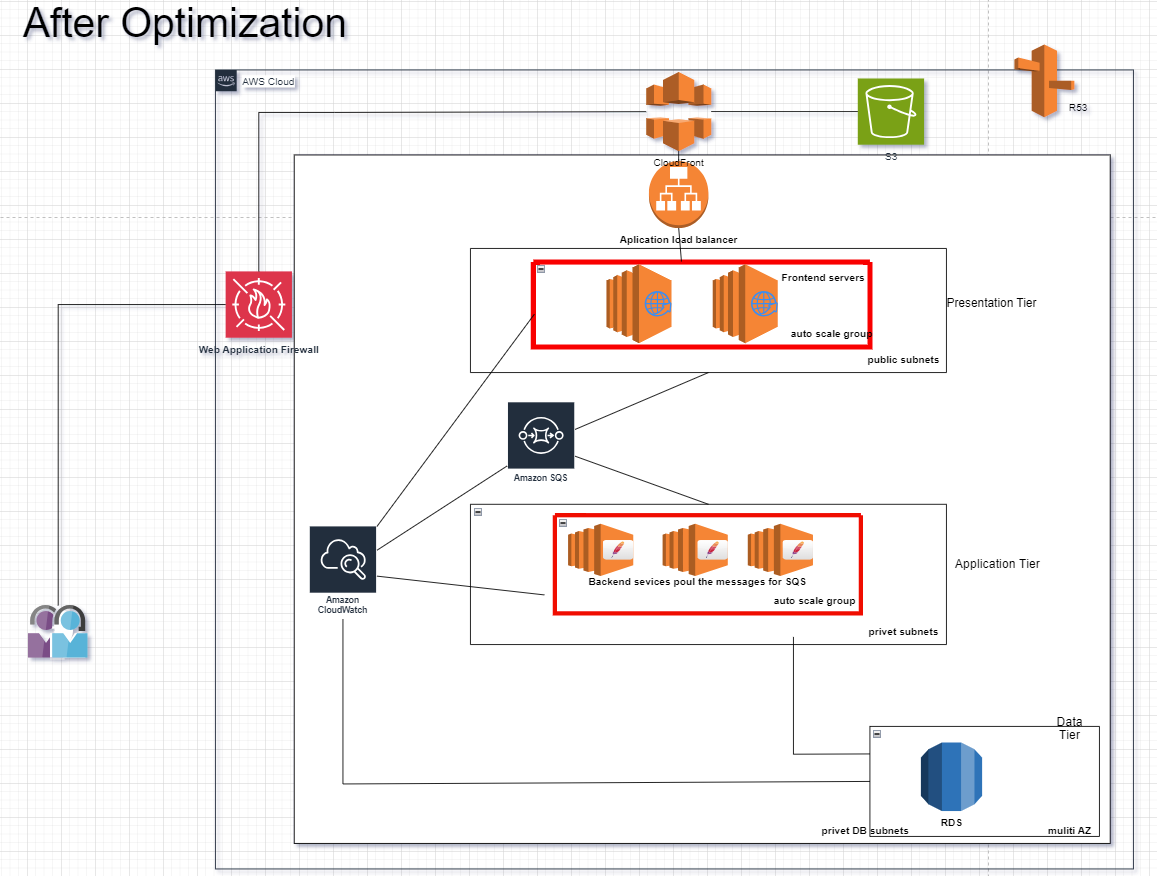


1. Create a CloudWatch Alarm  
     
   Navigate to alarms and create a New Alarm with a **"Select metric"**.  
     
   Eg: Set Alarm condition to trigger **Whenever cpu\_usage\_system is Greater/Equal to 80%** as below in the conditions section and setup.  
     
     
     
     
   

# **Optimization**

## **Resource Scaling**

Automatically scale resources (e.g., add/remove server instances) based on performance metrics.



Implemented Auto Scale group in Application Tire. This will cater the load according to the configured thresholds by add/remove server instances

On this case for adding /remove server instances based on the performance metrics (Eg: CPU usage ) we can use **Auto Scale Group**.

Also we can written all in the **AWS CloudFormation** and used to implement and manage infrastructure on AWS in an automated and consistent manner.

Steps:-

1. First we have to create “launch template” with the same configurations which we used to spin the earlier EC2 (we can add all the required installations/ configurations in “user data” section and crate this template).  
   Or  
   We can Create a AMI by using the already created/configured EC2 instance and use that AMI for the auto scale group.
2. Create an Auto Scaling Group with Choosing the **Launch Template** you created earlier.

Configure the Group Size and Scaling Policies:  
 Eg:-  
 Set the Desired Capacity: Enter 1 for the initial number of instances.  
 Minimum Capacity: Enter 1.  
 Maximum Capacity: Enter 4.

OR  
 Create a **CloudFormation** Script with using the created AMI ID and auto scale group.

Eg:-  
If we go with the **CloudFormation,** we can use the below code to implement the application tire infrastructure (cfdemo.yaml is a sample code and its written only to the application tire with **minimal of 1 instances and auto scale up to 4 instances** based on the **CPU performance** metrics and alarms. In this approach we can implement entire infastrctur by a **CloudFormation** script)   
CPUUtilizationHigh >= 70 (GreaterThanOrEqualToThreshold - CloudWatch Alarm for Scaling Out)

CPUUtilizationHigh >= 30 (LessThanThreshold - CloudWatch Alarm for Scaling In)

MinSize: 1

MaxSize: 4

cfdemo.yaml (<https://github.com/harshat85/MadMobileAssignment/blob/main/CFdemo.yaml>)

AWSTemplateFormatVersion: '2010-09-09'

Description: CloudFormation template to create an Auto Scaling Group in the Stockholm (eu-north-1) region.

Resources:

# Launch Configuration

MyAppLaunchConfiguration:

Type: AWS::AutoScaling::LaunchConfiguration

Properties:

ImageId: ami-064c5012ae6debf9b # this AMI is available in the eu-north-1 region

InstanceType: t3.micro # instance type

KeyName: ec2key # <-- Add the KeyName property

IamInstanceProfile: !Ref MyAppInstanceProfile

SecurityGroups:

- sg-09585339288ead35a

# IAM Instance Profile

MyAppInstanceProfile:

Type: AWS::IAM::InstanceProfile

Properties:

Roles:

- myApp\_Roll

# Auto Scaling Group

MyAppAutoScalingGroup:

Type: AWS::AutoScaling::AutoScalingGroup

Properties:

VPCZoneIdentifier:

- subnet-0e642b393b88a1507 # Update these with your actual subnet IDs

- subnet-0e3c4cca4b69c4039

LaunchConfigurationName: !Ref MyAppLaunchConfiguration

MinSize: 1 # minimum size

MaxSize: 4 # scale up to

DesiredCapacity: 1

HealthCheckType: EC2

HealthCheckGracePeriod: 300

MetricsCollection:

- Granularity: "1Minute"

Tags:

- Key: Name

Value: MyAppInstance

PropagateAtLaunch: true

# Scaling Policy - Scale Out (Increase Capacity)

ScaleOutPolicy:

Type: AWS::AutoScaling::ScalingPolicy

Properties:

AutoScalingGroupName: !Ref MyAppAutoScalingGroup

PolicyType: SimpleScaling

AdjustmentType: ChangeInCapacity

Cooldown: 300

ScalingAdjustment: 1

# Scaling Policy - Scale In (Decrease Capacity)

ScaleInPolicy:

Type: AWS::AutoScaling::ScalingPolicy

Properties:

AutoScalingGroupName: !Ref MyAppAutoScalingGroup

PolicyType: SimpleScaling

AdjustmentType: ChangeInCapacity

Cooldown: 300

ScalingAdjustment: -1

# CloudWatch Alarm for Scaling Out

CPUUtilizationHigh:

Type: AWS::CloudWatch::Alarm

Properties:

AlarmDescription: "Alarm if CPU Utilization is greater than or equal to 70%"

MetricName: CPUUtilization

Namespace: AWS/EC2

Statistic: Average

Period: 300

EvaluationPeriods: 1

Threshold: 70

ComparisonOperator: GreaterThanOrEqualToThreshold

Dimensions:

- Name: AutoScalingGroupName

Value: !Ref MyAppAutoScalingGroup

AlarmActions:

- !Ref ScaleOutPolicy

# CloudWatch Alarm for Scaling In

CPUUtilizationLow:

Type: AWS::CloudWatch::Alarm

Properties:

AlarmDescription: "Alarm if CPU Utilization is less than 30%"

MetricName: CPUUtilization

Namespace: AWS/EC2

Statistic: Average

Period: 300

EvaluationPeriods: 1

Threshold: 30

ComparisonOperator: LessThanThreshold

Dimensions:

- Name: AutoScalingGroupName

Value: !Ref MyAppAutoScalingGroup

AlarmActions:

- !Ref ScaleInPolicy

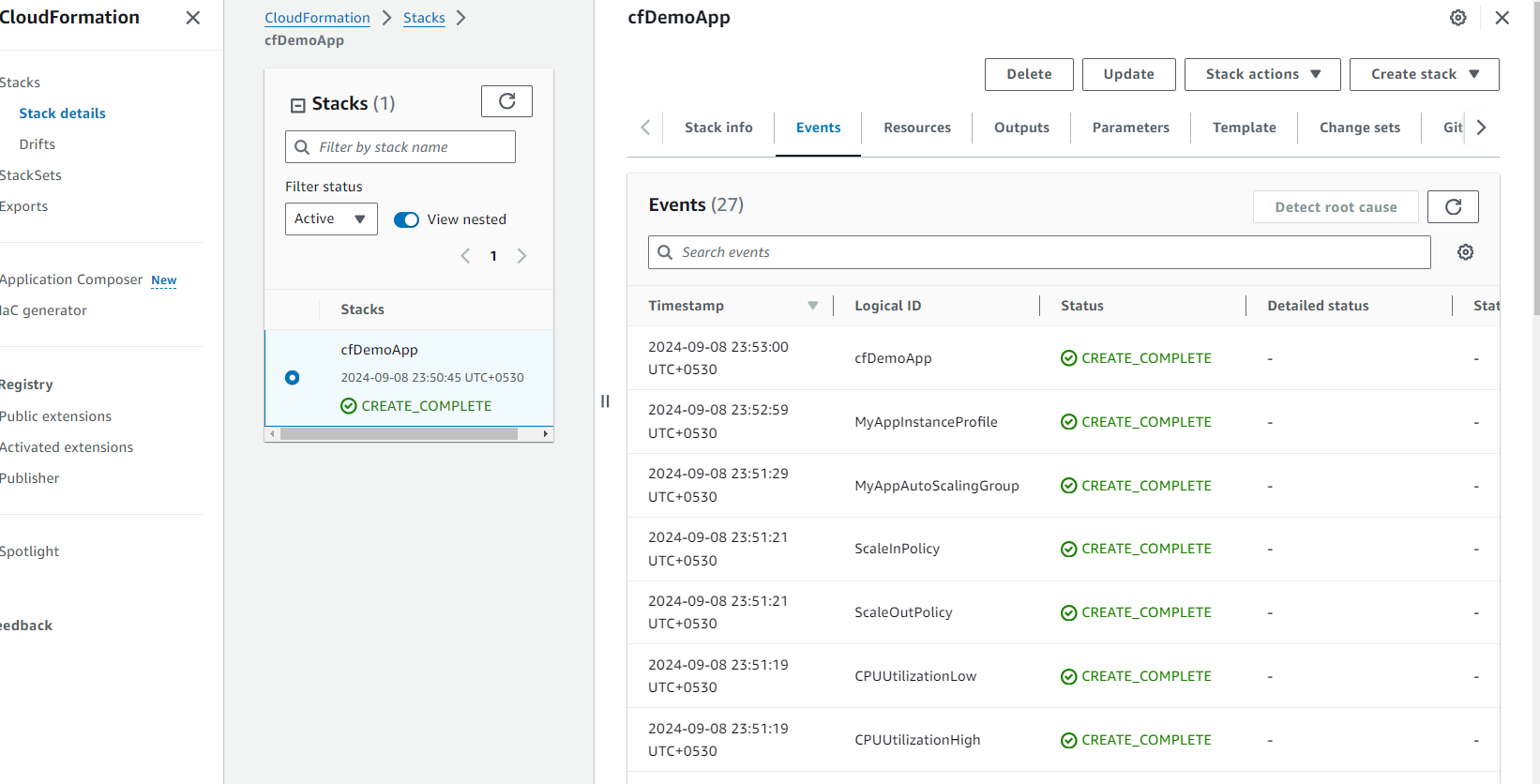
Outputs:

AutoScalingGroupName:

Description: "Name of the Auto Scaling Group"

Value: !Ref MyAppAutoScalingGroup

Test Sample in CF



# **Optional Enhancements**

## **Log Rotation**

Implement a log rotation script to manage log files, archiving or deleting old logs after a specified period.  
  
We can use the “logrotate.d” to get this done. To ensuring logs are regularly rotated, compressed, and cleaned up we can implement this.  
  
Create a file named /etc/logrotate.d/myapp (<https://github.com/harshat85/MadMobileAssignment/blob/main/myapp.txt>)

/var/log/myapp/\*.log {

daily # Rotate logs daily

rotate 7 # Keep only 7 rotations (one week of logs)

compress # Compress the rotated logs

delaycompress # Delay compression to the next rotation cycle

missingok # Ignore missing files and errors

notifempty # Do not rotate empty logs

dateext # Append date to rotated logs

dateformat -%Y-%m-%d # Specify date format in filenames

create 0644 root root # Specify permissions for new log files

olddir /var/log/myapp/archive # Archive rotated logs in a separate directory

postrotate

find /var/log/myapp/archive -mtime +90 -type f -delete # Delete logs older than 90 days

endscript

}

Configuration Logic:-  
  
Daily Rotation: Logs in /var/log/myapp/ are rotated daily.

File Naming: Rotated log files will have a date suffix (e.g., app.log-2024-09-08).

Compression: Logs older than the most recent rotation are compressed (e.g., app.log-2024-09-07.gz).

Archive Management: Rotated logs are moved to /var/log/myapp/archive.

Retention Policy: Only the last 7 days' worth of logs are kept in rotation.

Cleanup of Old Logs: Any log files older than 90 days in the archive directory are deleted to manage disk space.  
  
To Test the configurations manually.  
  
# sudo logrotate -f /etc/logrotate.d/myapp

**OR**

We can set this in a shell script like below.  
Create a configuration file named log\_rotation.conf to store parameters. (<https://github.com/harshat85/MadMobileAssignment/blob/main/log_rotation.conf>)

# log\_rotation.conf

# Directory containing log files

LOG\_DIR="/var/log/myapp"

# Directory to archive old logs

ARCHIVE\_DIR="/var/log/myapp/archive"

# Log file pattern

LOG\_PATTERN="\*.log"

# Number of days to retain logs

RETENTION\_DAYS=90

Shell script :- log\_rotation.sh(<https://github.com/harshat85/MadMobileAssignment/blob/main/log_rotation.sh>)

#!/bin/bash

# Load configuration

source /path/to/log\_rotation.conf

# Create archive directory if it does not exist

mkdir -p "$ARCHIVE\_DIR"

# Move rotated logs to archive and compress them

for log\_file in "$LOG\_DIR"/$LOG\_PATTERN; do

if [ -f "$log\_file" ]; then

# Move the log file to the archive directory with a date suffix

mv "$log\_file" "$ARCHIVE\_DIR/$(basename "$log\_file")-$(date +'%Y-%m-%d')"

# Compress the archived log file

gzip "$ARCHIVE\_DIR/$(basename "$log\_file")-$(date +'%Y-%m-%d')"

fi

done

# Delete archived logs older than $RETENTION\_DAYS days

find "$ARCHIVE\_DIR" -type f -mtime +$RETENTION\_DAYS -exec rm -f {} \;

To make the Script Executable  
# chmod +x /path/to/log\_rotation.sh  
  
Add this cron to schedule the script to run daily at midnight:  
# 0 0 \* \* \* /path/to/log\_rotation.sh

## **System Simulation**

Write a script to simulate peak load conditions and measure the system’s performance, allowing you to test the effectiveness of your optimizations.  
  
To this we can use Locust ( A scalable load testing tool written in Python.)  
  
Steps:-  
1. Install Locust: Install Locust using pip:

sudo yum install python3-pip -y  
pip install locust   
  
2. Create a Locust File: Create a file named locustfile.py

from locust import HttpUser, TaskSet, task, between

class UserBehavior(TaskSet):

@task

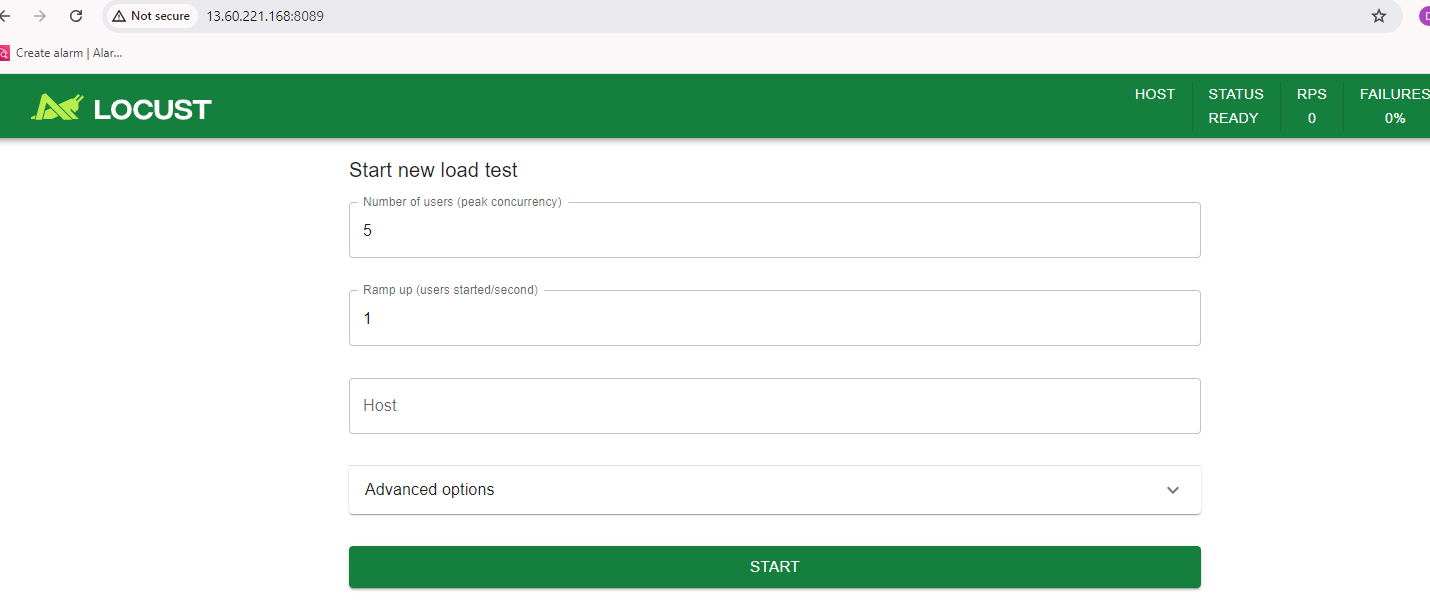
def get\_api(self):

self.client.get**("/your-api-endpoint")**

class WebsiteUser(HttpUser):

tasks = [UserBehavior]

wait\_time = between(1, 5) # Wait time between tasks

Execute Locust by  
  
# locust -f locustfile.py  
  
Open a browser and navigate to http://<IP>:8089 to configure and start the load test.  
(Need to configure inbound rule in security group allows traffic on port 8089)  
  
  
  
  
  
**OR**

In the optimization section, the proposed solution (automatically scaling resources by adding or removing server instances based on performance metrics) can be tested using a CPU stress test.  
  
  
Steps: -   
  
Run a CPU stress test to trigger scaling:-

# sudo yum install stress -y # For Amazon Linux or CentOS

# stress --cpu 2 --timeout 300 # Stress test for 5 minutes

Check the scaling actions in the **Auto Scaling Activity** section of the Auto Scaling group to see, instances are launching when CPU utilization exceeds the threshold.  
  
Auto Scaling group in AWS that automatically scales EC2 instances based on CPU utilization, with a starting capacity of 1 and a maximum of 4 instances. Once the CPU utilization decreases and stays low, the Auto Scaling group will automatically reduce the number of instances to match the lower load, scaling down to as few as 2 instances, according to your configuration.

## **Automated Deployment**

Develop a script to automate the deployment of the optimized system setup, making it easy to replicate in a different environment.

We can use **AWS CloudFormation** service to implement and manage infrastructure on AWS in an automated and consistent manner (infrastructure as code (IaC) using templates, which can be written in JSON or YAML format)  
We can provision the entire infrastructure stack, including VPC, subnets, EC2 instances for the web and application tiers, RDS for the database tier, and S3 buckets for static content.  
  
Below mentioned cfdemo.yaml is a sample template specifically designed for the application tier, with a minimum of 1 instance and the ability to automatically scale up to 4 instances based on CPU performance metrics and alarms. With this approach, we can deploy the entire infrastructure using a single CloudFormation script in a different environments.

GitHub Code link for the Sample CF code.  
<https://github.com/harshat85/MadMobileAssignment/blob/main/CFdemo.yaml> (**CFdemo.yaml**)

AWSTemplateFormatVersion: '2010-09-09'

Description: CloudFormation template to create an Auto Scaling Group in the Stockholm (eu-north-1) region.

Resources:

# Launch Configuration

MyAppLaunchConfiguration:

Type: AWS::AutoScaling::LaunchConfiguration

Properties:

ImageId: ami-064c5012ae6debf9b # this AMI is available in the eu-north-1 region

InstanceType: t3.micro # instance type

KeyName: ec2key # <-- Add the KeyName property

IamInstanceProfile: !Ref MyAppInstanceProfile

SecurityGroups:

- sg-09585339288ead35a

# IAM Instance Profile

MyAppInstanceProfile:

Type: AWS::IAM::InstanceProfile

Properties:

Roles:

- myApp\_Roll

# Auto Scaling Group

MyAppAutoScalingGroup:

Type: AWS::AutoScaling::AutoScalingGroup

Properties:

VPCZoneIdentifier:

- subnet-0e642b393b88a1507 # Update these with your actual subnet IDs

- subnet-0e3c4cca4b69c4039

LaunchConfigurationName: !Ref MyAppLaunchConfiguration

MinSize: 1 # minimum size

MaxSize: 4 # scale up to

DesiredCapacity: 1

HealthCheckType: EC2

HealthCheckGracePeriod: 300

MetricsCollection:

- Granularity: "1Minute"

Tags:

- Key: Name

Value: MyAppInstance

PropagateAtLaunch: true

# Scaling Policy - Scale Out (Increase Capacity)

ScaleOutPolicy:

Type: AWS::AutoScaling::ScalingPolicy

Properties:

AutoScalingGroupName: !Ref MyAppAutoScalingGroup

PolicyType: SimpleScaling

AdjustmentType: ChangeInCapacity

Cooldown: 300

ScalingAdjustment: 1

# Scaling Policy - Scale In (Decrease Capacity)

ScaleInPolicy:

Type: AWS::AutoScaling::ScalingPolicy

Properties:

AutoScalingGroupName: !Ref MyAppAutoScalingGroup

PolicyType: SimpleScaling

AdjustmentType: ChangeInCapacity

Cooldown: 300

ScalingAdjustment: -1

# CloudWatch Alarm for Scaling Out

CPUUtilizationHigh:

Type: AWS::CloudWatch::Alarm

Properties:

AlarmDescription: "Alarm if CPU Utilization is greater than or equal to 70%"

MetricName: CPUUtilization

Namespace: AWS/EC2

Statistic: Average

Period: 300

EvaluationPeriods: 1

Threshold: 70

ComparisonOperator: GreaterThanOrEqualToThreshold

Dimensions:

- Name: AutoScalingGroupName

Value: !Ref MyAppAutoScalingGroup

AlarmActions:

- !Ref ScaleOutPolicy

# CloudWatch Alarm for Scaling In

CPUUtilizationLow:

Type: AWS::CloudWatch::Alarm

Properties:

AlarmDescription: "Alarm if CPU Utilization is less than 30%"

MetricName: CPUUtilization

Namespace: AWS/EC2

Statistic: Average

Period: 300

EvaluationPeriods: 1

Threshold: 30

ComparisonOperator: LessThanThreshold

Dimensions:

- Name: AutoScalingGroupName

Value: !Ref MyAppAutoScalingGroup

AlarmActions:

- !Ref ScaleInPolicy

Outputs:

AutoScalingGroupName:

Description: "Name of the Auto Scaling Group"

Value: !Ref MyAppAutoScalingGroup

# Task 2: Process Improvement and Communication

# **Requirements**

## **Process Review**

Assumptions: Assuming the current incident management process is implemented as described below.

### **Weakness Identification: Analyze the current incident management process, identifying key weaknesses (e.g., lack of automation, communication gaps, role ambiguities).** Based on the implemented incident management process, the following points can be highlighted as key weaknesses.

Lack of Automation:   
It involves the usage of several manual interfaces which result in increased time, cost and high probability of errors. For instance, reporting of events like logging incidents, distribution of incidents to the right teams/sections, and concern arising from escalation among others. This handling of materials slows down the process, and brings in errors, which leads to delays.

Communication Gaps:  
But still, there are no common platforms for exchanging information regarding particular incidents.

Intersector/inter team or with the stakeholders, communication may not be effective.

Eg:- while the IT support team might be handling incidents, the end-users or other teams need not be aware of the progress in real-time.

This may result into conflicts between team members and stakeholders, unnecessary repeat of tasks, and slow problem solving.

Unclear Role Assignments:  
The process does not contain clear division of the parties, which results into confusion whenever there is an incident. There must be some rules of how the certain incidents are solved or escalated when necessary; if not, incidents may remain unresolved or proceed with no proper solution for a longer period than needed.

Incident Classification:   
The classification of an incident is still done by hand, and may not be classified correctly, this affects the amount of time it takes to resolve an incident.

Limited Visibility:   
The process also does not give the real-time status on the incidents and how they are being resolved this in turn makes it hard to see the time taken to resolve incidents and possible areas which need to be improved.

Lack of Automation:  
Automation could be made to reduce a number of the conventional tasks, for example, automatically categorizing or routing the incidents as per some pre-defined criteria or moving the incidents to another higher level after they have taken certain amount of time.

### **Improvement Proposal:**

**Suggest specific improvements to the process, focusing on automation, role clarity, and enhanced communication**

Automate Incident Triage and Assignment:   
Automation ensures incidents are assigned faster and to the correct team without human intervention.  
criteria (eg:- incident type, severity, affected systems).  
Integrate monitoring tools with the incident management tool to automatically create incident tickets, removing the need for manual reporting.  
Implement a rules-based engine to automatically assign incidents to team members according to their expertise, availability, and workload.  
To Automate Incident Status Updates we can implement automated workflows to update incident status in real-time, reducing manual intervention and minimizing errors.  
Eg: Automated notification workflows, Automated escalation rules, Real-time incident status updates

Implement Clear Role Definition:   
It is crucial to establish and put down clearly the roles and responsibilities of each and every team or individual taking part in the incident management process. This will eliminate confusion in relation to management responsibility in regard to each of the described process steps, including escalation of incidents and their resolution.

When it comes to the implementation of steps associated with incident management, it is suggested to construct a RACI matrix which will help to define who is responsible for which activity.

Proactively supply the proper training and tools to guarantee the members of the team know what they are supposed to do.  
  
Use Communication Platforms:  
Eg:- Slack/MS Teams  
Incident responsive platform shall be synchronized with real-time communication products such as Slack and Microsoft teams. Make sure that everyone remains updated in real-time without having to use follow-ups whenever an incident is logged, or escalated by this tool.

Follow-ups and reminders to stakeholders/ Team members not only for frequent updates on the incidents but also do the frequent reminders.

The next action should be to draw up a Communication Plan that identifies the general and specific communication/s, schedules and means to be used when passing on updates to the on-going incidents.

Conduct Periodic Reviews:   
It is necessary to set some frequency (weekly or monthly) to review the incident management process and define with which aspects there are problems. These reviews can check things such as how many cases reached a higher level, how long cases took to be resolved, and in what ways communication failed.

With the help of utilizing a knowledge base, storing procedures and results of the incidents and potential causes with feedbacks regarding how to avoid such type of incidents in the future.

### **Success Metrics:** **Define metrics to measure the success of the proposed improvements (e.g., reduced incident resolution time, improved response times).**

Incident Resolution Time:

Mean Time to Resolve: Measure the average time taken to resolve incidents, goal is to reduce MTTR by 20% within the next 3 months.  
Eg:- Target: 2.8 hours, Current: 4 hours, Improvement: 30%

Incident Resolution Rate: Track the percentage of incidents resolved within a specified timeframe (e.g., 80% of incidents resolved within 2 hours).  
Eg:- Target: 85%, Current: 70%, Improvement: 21%

Response Times:

Mean Time to Response: Measure the average time taken to respond to incidents, from the moment an incident is logged to the time a team member begins working on it. Goal is to reduce the average Time to respond by 50% within the next 3 months.  
Eg:- Target: 15 minutes, Current: 30 minutes, Improvement: 50%  
  
Response Rate: Track the percentage of incidents responded to within a specified timeframe (e.g., 90% of incidents responded to within 30 minutes).  
Eg:- Target: 92%, Current: 85%, Improvement: 8.2%

Automation and Efficiency:

Automation Rate: Measure the percentage of incidents automatically reported and assigned, Automating the incident assignment process should allow incidents to be assigned to the appropriate team/unit within five minutes of being logged, ensuring no unnecessary delays in the resolution process.  
Goal is to increase automation rate from 20% to 40% within the next 6 months.  
Eg:- Target: 40%, Current: 20%, Improvement: 100%  
  
Manual Intervention Rate: Track the percentage of incidents requiring manual intervention, Goal is to reduce manual intervention rate from 60% to 45% within the next 3 months.  
Eg:- Target: 45%, Current: 60%, Improvement: 25%

Communication and Collaboration:

Communication Satisfaction: Measure stakeholder satisfaction with incident communication, Goal is to increase satisfaction ratings from 80% to 92% within the next 6 months.  
Eg:- Target: 92%, Current: 80%, Improvement: 15%  
  
Collaboration Index: Track the frequency and quality of collaboration between team members, Goal is to increase collaboration index from 6 to 7.5 (on a scale of 1-10) within the next 3 months.  
Eg:- Target: 7.5, Current: 6, Improvement: 25%  
  
Escalated Incidents Due to Communication Delays: The overall number of customers escalations should be lower, when it comes to escalated cases caused by communication problems – it should be fewer. Lesser incidents will be escalated as the status update is automated and the team members are always updated in real-time.

Knowledge Management:

“Knowledge Base” Utilization: Measure the frequency of knowledge base usage, Goal is to increase utilization from 50% to 65% within the next 6 months.  
Eg:- Target: 65%, Current: 50%, Improvement: 30%

Repeat Incident Rate: Track the percentage of repeat incidents, Goal is to reduce repeat incident rate from 15% to 12% within the next 3 months.  
Eg:- Target: 12%, Current: 15%, Improvement: 20%

Process Maturity:

Process Compliance: Measure the percentage of incidents following the defined incident management process, aiming to increase compliance from 80% to 90% within the next 6 months.  
Eg:- Target: 90%, Current: 80%, Improvement: 12.5%  
  
Process Improvement Rate: Track the frequency of process improvements, aiming to increase process improvement rate from 2 to 3 improvements per quarter within the next 3 months.  
Eg:- Target: 3, Current: 2, Improvement: 50%

## **Automation**

### **Incident Automation Script:**

#### **Auto-Assignment: A script that automatically assigns incidents to the appropriate team based on predefined criteria.**

Assuming the below data set is the Sample Data Set:-

Incident 1: Network incident with High severity in NYC

Incident 2: Server incident with Critical severity in LON

Incident 3: Security incident with Normal severity in LKA

Incident 4: Network incident with High severity in NYC

Incident 5: Server incident with Critical severity in LON

Incident 6: Security incident with Normal severity in IND

Incident 7: Network incident with High severity in AUS

Incident 8: Server incident with Critical severity in NYC

Incident 9: Security incident with Normal severity in LKA

Incident 10: Network incident with High severity in LON

[

{

"id": 1,

"type": "Network",

"severity": "High",

"location": "NYC"

},

{

"id": 2,

"type": "Server",

"severity": "Critical",

"location": "LON"

},

{

"id": 3,

"type": "Security",

"severity": "Normal",

"location": "LKA"

},

{

"id": 4,

"type": "Network",

"severity": "High",

"location": "NYC"

},

{

"id": 5,

"type": "Server",

"severity": "Critical",

"location": "LON"

},

{

"id": 6,

"type": "Security",

"severity": "Normal",

"location": "IND"

},

{

"id": 7,

"type": "Network",

"severity": "High",

"location": "AUS"

},

{

"id": 8,

"type": "Server",

"severity": "Critical",

"location": "NYC"

},

{

"id": 9,

"type": "Security",

"severity": "Normal",

"location": "LKA"

},

{

"id": 10,

"type": "Network",

"severity": "High",

"location": "LON"

}

]

IncidentAutomation.sh

#!/bin/bash

# Define API endpoint and authentication details

API\_ENDPOINT="https://incident-management-api.com/incidents"

USERNAME="your\_username"

PASSWORD="your\_password"

# Define criteria for auto-assignment

CRITERIA\_INCIDENT\_TYPE=("Network" "Server" "Security")

CRITERIA\_SEVERITY=("High" "Critical" "Normal")

CRITERIA\_LOCATION=("NYC" "LON" "LKA" "IND" "AUS")

# Define teams and their corresponding criteria

TEAMS=("Network Team" "Server Team" "NYC Team" "LON Team" "LKA Team" "IND Team" "AUS Team")

TEAM\_CRITERIA=("incident\_type=Network" "incident\_type=Server" "incident\_type=Security" "location=NYC" "location=LON" "location=LKA" "location=IND" "location=AUS")

# Authenticate with the API

AUTH=$(echo -n "$USERNAME:$PASSWORD" | base64)

# Get the list of incidents

INCIDENTS=$(curl -X GET \

$API\_ENDPOINT \

-H 'Authorization: Basic '$AUTH \

-H 'Content-Type: application/json')

# Loop through each incident

for INCIDENT in $(echo $INCIDENTS | jq -r '.[] | @uri'); do

# Extract incident details

INCIDENT\_ID=$(echo $INCIDENT | jq -r '.id')

INCIDENT\_TYPE=$(echo $INCIDENT | jq -r '.type')

SEVERITY=$(echo $INCIDENT | jq -r '.severity')

LOCATION=$(echo $INCIDENT | jq -r '.location')

# Check if the incident meets the auto-assignment criteria

if [[ " ${CRITERIA\_INCIDENT\_TYPE[@]} " =~ " $INCIDENT\_TYPE " ]] && \

[[ " ${CRITERIA\_SEVERITY[@]} " =~ " $SEVERITY " ]] && \

[[ " ${CRITERIA\_LOCATION[@]} " =~ " $LOCATION " ]]; then

# Determine the team to assign the incident to

for ((i=0; i<${#TEAMS[@]}; i++)); do

if eval "[[ \$INCIDENT\_TYPE == ${TEAM\_CRITERIA[$i]} ]]"; then

ASSIGNED\_TEAM=${TEAMS[$i]}

break

fi

done

# Assign the incident to the team

ASSIGNMENT\_ENDPOINT="$API\_ENDPOINT/$INCIDENT\_ID/assign"

curl -X PATCH \

$ASSIGNMENT\_ENDPOINT \

-H 'Authorization: Basic '$AUTH \

-H 'Content-Type: application/json' \

-d '{"team": "'$ASSIGNED\_TEAM'"}'

# Check if the assignment was successful

if [ $? -eq 0 ]; then

echo "Assigned incident $INCIDENT\_ID to $ASSIGNED\_TEAM"

else

echo "Error assigning incident $INCIDENT\_ID"

fi

fi

done

#### **Incident Escalation: A script that escalates incidents based on severity and elapsed time since the incident was logged.**

IncidentEscalation.sh

import json

import datetime

import time

# Define the incident data

incidents = [

{"id": 1, "type": "Network", "severity": "High", "location": "NYC", "logged\_at": "2022-01-01 10:00:00"},

{"id": 2, "type": "Server", "severity": "Critical", "location": "LON", "logged\_at": "2022-01-01 11:00:00"},

{"id": 3, "type": "Security", "severity": "Normal", "location": "LKA", "logged\_at": "2022-01-01 12:00:00"},

{"id": 4, "type": "Network", "severity": "High", "location": "NYC", "logged\_at": "2022-01-01 13:00:00"},

{"id": 5, "type": "Server", "severity": "Critical", "location": "LON", "logged\_at": "2022-01-01 14:00:00"},

{"id": 6, "type": "Security", "severity": "Normal", "location": "IND", "logged\_at": "2022-01-01 15:00:00"},

{"id": 7, "type": "Network", "severity": "High", "location": "AUS", "logged\_at": "2022-01-01 16:00:00"},

{"id": 8, "type": "Server", "severity": "Critical", "location": "NYC", "logged\_at": "2022-01-01 17:00:00"},

{"id": 9, "type": "Security", "severity": "Normal", "location": "LKA", "logged\_at": "2022-01-01 18:00:00"},

{"id": 10, "type": "Network", "severity": "High", "location": "LON", "logged\_at": "2022-01-01 19:00:00"}

]

# Define the escalation rules

escalation\_rules = {

"Critical": {"time\_threshold": 30, "escalation\_level": "SEV1"},

"High": {"time\_threshold": 60, "escalation\_level": "SEV2"},

"Normal": {"time\_threshold": 120, "escalation\_level": "SEV3"}

}

# Iterate over the incidents and check for escalation

for incident in incidents:

id = incident["id"]

severity = incident["severity"]

logged\_at = datetime.datetime.strptime(incident["logged\_at"], "%Y-%m-%d %H:%M:%S")

elapsed\_time = (datetime.datetime.now() - logged\_at).total\_seconds() / 60

time\_threshold = escalation\_rules[severity]["time\_threshold"]

escalation\_level = escalation\_rules[severity]["escalation\_level"]

if elapsed\_time > time\_threshold:

print(f"Escalating incident {id} to {escalation\_level} after {elapsed\_time:.2f} minutes")

else:

print(f"Incident {id} is still within the time threshold ({elapsed\_time:.2f} minutes)")

1. The incident data is stored in a list of dictionaries,  
   each dictionary represents an incident with its ID, type, severity, location, and logged time
2. The escalation rules are defined in a dictionary.  
   “time\_threshold” and “escalation\_level”
3. The script iterates over the incidents and extracts the relevant information using dictionary lookups.
4. The script calculates the elapsed time since the incident was logged using ‘datetime’ module.
5. The script checks if the elapsed time exceeds the time threshold for the incident's severity, and prints a message indicating whether the incident should be escalated.  
     
   Note :- assumes that datetime module is available

#### **Integration: Describe how this script could be integrated into the existing incident management system.**

The Python script can be integrated into the existing incident management system to automate the escalation process.

API Integration - The script can be integrated with the incident management system's API to retrieve incident data and update the escalation status. The API can provide the necessary incident data, such as ID, type, severity, location, and logged time, which can be used by the script to determine the escalation status.

The script receives the incident data from the API and uses it to determine whether the incident needs to be escalated or not. This is done by applying the escalation rules defined in the script.  
  
If the script trigger that an incident needs to be escalated, it sends an update request to the API to change the escalation status of the incident. This ensures that the incident management system is aware of the escalation and can take further action.

#### **Optional Enhancements:**

To streamline the incident management process, we can integrate additional tools like chatbots and notification systems with the existing incident management system.

Notification System Integration  
By integrating a notification system, we can send automated alerts and updates to team members, stakeholders, customers, and management. This ensures that everyone is informed and up-to-date on the incident status.

Chatbot Integration  
Chatbots can be integrated to provide a centralized communication platform.  
- Receive incident notifications and updates  
- Assign tasks and track progress

#### **Communication and Training Plan for New Incident Management Processes:**

A communication and training plan will be necessary to communicate the new incident management processes. Such a plan will make it easier to integrate the new tools and processes seamlessly, decrease disruptions & ensure all stakeholders are aware & prepared enough to handle incidents with equal effectiveness.

**Communication Plan:**

Goal: Broadly educate and excite stakeholders of new incident management processes and tools.  
  
**Target Audience:**

# Incident management team members

# IT staff

# Customer-facing teams

# Management

# Stakeholders

**Communication Channels:**

# Email notifications

# Team meetings

# Training sessions

# Intranet announcements

# Digital signage

**Communication Timeline:**

# 2 week before Launching: Send a stakeholders all-hands with introductory information to the incident management processes and tools.

# 1 week before Launching: Hold team meetings to train and explain the new processes / tools.

# Announce launch day: Send a communication to all relevant stakeholders that the new incident management system is now available with details of its benefits and features.

# After-launch: Continue to provide system updates and usage tips in regular communications with the stakeholders.

**Training Proposal:**

Goal: Train stakeholders on the new incident management tools & processes, so they can utilize them effectively.

**Training Modules:**

Members of the incident management team:

Hands-on training on using the new tools and processes.

More comprehensive training on the new system with features like API integration, chatbot integration and notification systems.

IT staff:

# Training on the new system — how to use it, integrating APIs and configuration.

# Teaching how to troubleshoot and about the technical stuff

Customer-facing teams:

# New incident management processes and how customer communications are trained

# New system to brief customers about Incident meant training

Management and stakeholders:

# Training on the new Incident Management System and its advantages across all key interfaces

# Training on how to use the system for incident status tracking and communication.

Training Methods:

# Guided or instructor-led training

# Online tutorials and videos

# Sittings of practice and simulation

# Practical training and coaching  
  
Training Timeline:

# 1 Week before Launch Train the incident management team and IT staff

# Go-Live: Conduct training for customer service and management teams.

# Offer continued training and support after launch to ensure that users can feel comfortable with the new system.

The above defined communication and training plan will help in transition process for new incident management processes & tools as well trained the stakeholders with required skills to use system.