### DOCUMENTATION ON BASIC SCRIPTING TASK 1. Write a Shell Script to Monitor Server Resource Usage

**Goal:** Monitor CPU, memory, and disk usage and send an email alert if thresholds are exceeded.

**Steps:**

**a) Set Up an AWS EC2 Instance**

1. **Launch an EC2 Instance**: Choose an Amazon Linux or Ubuntu AMI.
2. **Connect via SSH**: Use your private key to SSH into the instance.

**b) Install Necessary Tools**

* Install mailx for sending emails:

# For Ubuntu

#### Set up your SMTP configuration in /etc/mail.rc. c) Create the Shell Script

* Create a script (e.g., resource\_monitor.sh):

#!/bin/bash

THRESHOLD=90

EMAIL="sakshimhaske02@gmail.com"

LOG\_FILE="/var/log/resource\_monitor.log"

while true; do

CPU=$(top -bn1 | grep "Cpu(s)" | awk '{print $2}')

MEM=$(free | grep Mem | awk '{print $3/$2 \* 100.0}')

DISK=$(df / | grep / | awk '{print $5}' | sed 's/%//')

if (( $(echo "$CPU > $THRESHOLD" | bc -l) )); then

echo "CPU usage is above $THRESHOLD% ($CPU%)" | mail -s "CPU Alert" $EMAIL

fi

if (( $(echo "$MEM > $THRESHOLD" | bc -l) )); then

echo "Memory usage is above $THRESHOLD% ($MEM%)" | mail -s "Memory Alert" $EMAIL

fi

if (( DISK > THRESHOLD )); then

echo "Disk usage is above $THRESHOLD% ($DISK%)" | mail -s "Disk Alert" $EMAIL

fi

sleep 60

done  
  
  
\*\* Give execution permissions to monitor\_resources.sh   
  
 chmod +x monitor\_resources.sh  
  
\*\* Execute the file  
  
./ monitor\_resources.sh

**If using 2FA (Recommended)**:

* Go to **"App Passwords"** in your Google Account.
* Generate an app password for "Mail."
* Use this app password instead of your regular Gmail password.

Yes, you can use your Gmail account to send email alerts, but you’ll need to configure your EC2 instance or server to use Gmail’s SMTP server. Follow these steps:

**Step 1: Enable "Less Secure Apps" or Use App Passwords**

1. **If using your Google Account directly**:
   * Go to your Google Account Security Settings.
   * Enable **"Allow less secure apps"** (deprecated for most users, only available if you don't have 2FA enabled).
2. **If using 2FA (Recommended)**:
   * Go to **"App Passwords"** in your Google Account : snyz jjju gjou avrz
   * Generate an app password for "Mail."
   * Use this app password instead of your regular Gmail password.

**Step 2: Install mailutils or sendmail**

Install the email-sending utilities:

* On Ubuntu:

sudo apt-get update

sudo apt-get install mailutils -y

**Step 3: Configure Gmail SMTP**

1. Edit the configuration file for mailutils :

sudo nano /etc/mail.rc

1. Add the following configuration for Gmail:

set smtp=smtp://smtp.gmail.com:587

set smtp-auth-user=sakshimhaske02@gmail.com

set smtp-auth-password=snyz jjju gjou avrz (app password)

set smtp-auth=login

set from=sakshimhaske02@gmail.com

1. Save the file and exit.

### ****Step 4: Test Email****

Send a test email to verify the configuration:

echo "Test email body" | mail -s "Test Email Subject" sakshimhaske02@gmail.com

**Firewall Blocking Port 587**:

* Check your EC2 instance's **Security Group** rules. Add an outbound rule for port **587** if missing

#### 1. Update Your Postfix Configuration

Postfix is trying to connect on port 25. Reconfigure it to use Gmail's authenticated SMTP server with port **587**.

Edit the Postfix configuration file:

sudo nano /etc/postfix/main.cf

Add or update the following lines:

relayhost = [smtp.gmail.com]:587

smtp\_sasl\_auth\_enable = yes

smtp\_sasl\_password\_maps = hash:/etc/postfix/sasl\_passwd

smtp\_sasl\_security\_options = noanonymous

smtp\_tls\_security\_level = encrypt

smtp\_tls\_CAfile = /etc/ssl/certs/ca-certificates.crt

#### 2. Configure SASL Authentication

Create a file for your Gmail credentials:

sudo nano /etc/postfix/sasl\_passwd

Add the following:

[smtp.gmail.com]:587 sakshimhaske02@gmail.com: snyz jjju gjou avrz

After modifying /etc/postfix/sasl\_passwd, run the following command to update the password map:

sudo postmap /etc/postfix/sasl\_passwd

Also, restart the Postfix service:

sudo systemctl restart postfix

### 2. ****Stress Test with Multiple Processes****:

Sometimes, a single instance of yes or a loop may not show enough stress, especially if your CPU has multiple cores. To generate visible load, you can run multiple instances of a stress-inducing process.

1. **CPU Stress:  
     
   Run Multiple yes Processes**: Run multiple instances of the yes command:

yes > /dev/null &

yes > /dev/null &

yes > /dev/null &

This should generate significant CPU usage on multiple cores. You can monitor this with top or htop.  
  
To stop or kill the CPU stress process you initiated with the yes > /dev/null & command, follow these steps:

### 1.1 ****Find the Process IDs (PIDs)****:

You can use the ps or top command to find the PIDs of the running yes processes.

To find the PIDs of the yes commands running in the background, you can use:

ps aux | grep 'yes > /dev/null'

This will list the running yes commands and their PIDs. The output will look something like:

user 12345 0.0 0.0 1234 5678 pts/0 S 12:34 0:00 yes > /dev/null

user 12346 0.0 0.0 1234 5678 pts/0 S 12:34 0:00 yes > /dev/null

user 12347 0.0 0.0 1234 5678 pts/0 S 12:34 0:00 yes > /dev/null

In this case, the PIDs of the yes processes are 12345, 12346, and 12347.

### 1.2. ****Kill the Processes****:

Once you've identified the PIDs, you can kill the processes using the kill command:

kill 12345 12346 12347

Alternatively, you can kill all yes commands at once using:

pkill yes   
  
 **2.  Memory Stress**:

dd if=/dev/zero of=/dev/null bs=1M count=5000 &

To remove - kill $(pgrep dd)

**3.  Disk Stress**:

dd if=/dev/zero of=/tmp/testfile bs=1M count=500

To remove - rm /tmp/testfile

### ****Set Up a Cron Job:**** To monitor resources every 5 minutes for 1 hour, create a temporary cronjob.

1. **Edit the Cron Table**Open the Cron editor for the current user:  
     
   crontab -e

**Add the Cron Job**  
Add the following entry to execute the script every 5 minutes for 1 hour. Replace /home/ubuntu/resource\_monitor.sh with the actual path of the script.  
  
\*/5 0 \* \* \* /home/ubuntu/monitor\_resources.sh && sleep 1800 && pkill -f "/home/ubuntu/monitor\_resources.sh"

### ****Monitor the Logs:****

After 1 hour, review the resource monitoring logs:

cat /var/log/resource\_monitor.log

### CPU Uilization : Memory Usage : Disk Usage : Steps for the Python Task: Monitoring and Blocking Failed Login Attempts

**Objective:**  
Develop a Python script to monitor web server logs, identify failed login attempts, and dynamically block offending IP addresses using firewall rules.

### Step 1: Set Up the Environment

1. **Ensure Prerequisites**:
   * Python 3 is installed.
   * Install wordpress.
   * Access to server logs (e.g., Apache logs in /var/log/apache2/access.log).
   * Install wordpress plugin : Fail2ban (to monitor unknown user in auth.log)
   * Appropriate permissions to modify firewall rules (requires sudo).
2. **Install Required Python Libraries**: If additional libraries like re or subprocess are needed:

pip install re

pip install subprocess

### ****Step 2: Create the Python Script****

import re  
import subprocess  
LOG\_FILE = "/var/log/auth.log"  # Path to the log file  
THRESHOLD = 5  # Number of failed attempts before blocking  
BLOCKED\_IPS\_FILE = "/var/log/blocked\_ips.txt"  # File to keep track of blocked Ips  
def get\_failed\_login\_ips():  
    """Parse the log file and return a dictionary of IPs with failed login attempts."""  
    failed\_ips = {}  
    with open(LOG\_FILE, "r") as file:  
        for line in file:  
            ip = line.split()[0]  # Extract IP address            # Detect failed login attempts due to "unknown user"  
            if "wp-login.php" in line and "unknown user" in line:  
                failed\_ips[ip] = failed\_ips.get(ip, 0) + 1            # Additional checks for "invalid username"  
            elif "wp-login.php" in line and "invalid username" in line:  
                failed\_ips[ip] = failed\_ips.get(ip, 0) + 1    return failed\_ipsdef block\_ip(ip):  
    """Block an IP using iptables."""  
    try:  
        subprocess.run(["sudo", "iptables", "-A", "INPUT", "-s", ip, "-j", "DROP"], check=True)  
        print(f"Blocked IP: {ip}")  
        with open(BLOCKED\_IPS\_FILE, "a") as file:  
            file.write(ip + "\n")  
    except subprocess.CalledProcessError as e:  
        print(f"Error blocking IP {ip}: {e}")def main():  
    """Main function to monitor logs and block offending IPs."""  
 failed\_ips = get\_failed\_login\_ips()    # Read already blocked IPs to avoid reblocking  
    try:  
        with open(BLOCKED\_IPS\_FILE, "r") as file:  
            blocked\_ips = set(file.read().splitlines())  
    except FileNotFoundError:  
        blocked\_ips = set()    # Check and block IPs exceeding the threshold  
    for ip, count in failed\_ips.items():  
        if count >= THRESHOLD and ip not in blocked\_ips:  
            block\_ip(ip)if \_\_name\_\_ == "\_\_main\_\_":  
    main()  
  
\*\* Change Permissions   
chmod +x block\_failed\_logins.py

### ****Step 3: Perform failed login attempts to wordpress website by giving wrong username and password****

1. Run the script manually to check its functionality:

sudo python3 block\_failed\_logins.py

1. Verify that offending IPs are blocked using:

sudo iptables -L

### ****Step 4: Automate the Task with a Cron Job****

1. Open the crontab editor:

crontab -e

1. Add the following line to run the script every 5 minutes:  
     
   \*/5 \* \* \* \* /usr/bin/python3 /home/ubuntu/block\_failed\_logins.py  
     
   Save and exit.

