Placement Empowerment Program

Cloud Computing and DevOps Centre

Write a Python Script to Monitor an Application : Create a Python script that sends periodic HTTP requests to your application and alerts you if it's down.

Name: Nantha Krishnan.G.K.
Department: AML

Introduction

Ensuring high availability and reliability of an application is crucial for maintaining a seamless user experience. This Proof of Concept (PoC) focuses on developing a lightweight Python script to monitor the health status of an application by sending periodic HTTP requests.

Overview

Python Script to Monitor an Application

1. **Setting Up the Monitoring Environment**: Install Python and required libraries (requests, smtplib), and configure an email account for alerts.
2. **Making Periodic HTTP Requests**: Use requests to periodically send HTTP requests to the application URL.
3. **Defining Success and Failure Conditions**: Check HTTP response status codes (200 for success, non-200 for failure) to detect application status.
4. **Sending Email Alerts**: If the application is down, use smtplib to send an email alert to a specified recipient.
5. **Automating Periodic Checks**: Use an infinite loop (while True) to repeatedly check the application's status at regular intervals (e.g., every 60 seconds).
6. **Logging and Handling Errors**: Log any errors or failures and handle exceptions to ensure script reliability.

Objectives

Python Script to Monitor an Application

1. **Understanding Web Monitoring Fundamentals**: Learn how to periodically check the availability of web applications using HTTP requests.
2. **Practical Scripting Skills**: Gain hands-on experience in writing Python scripts that interact with web services and handle errors.
3. **Automated Alerting**: Develop a script that automatically detects application downtime and sends email notifications.
4. **Handling HTTP Status Codes**: Learn how to interpret HTTP status codes (e.g., 200, 404, 500) to assess the health of an application.

5. **Email Automation**: Gain experience in automating email alerts via SMTP to notify administrators when issues are detected.
6. **Improving Reliability**: Explore how to build a simple and reliable monitoring system that runs continuously to ensure your application is always available.

Importance

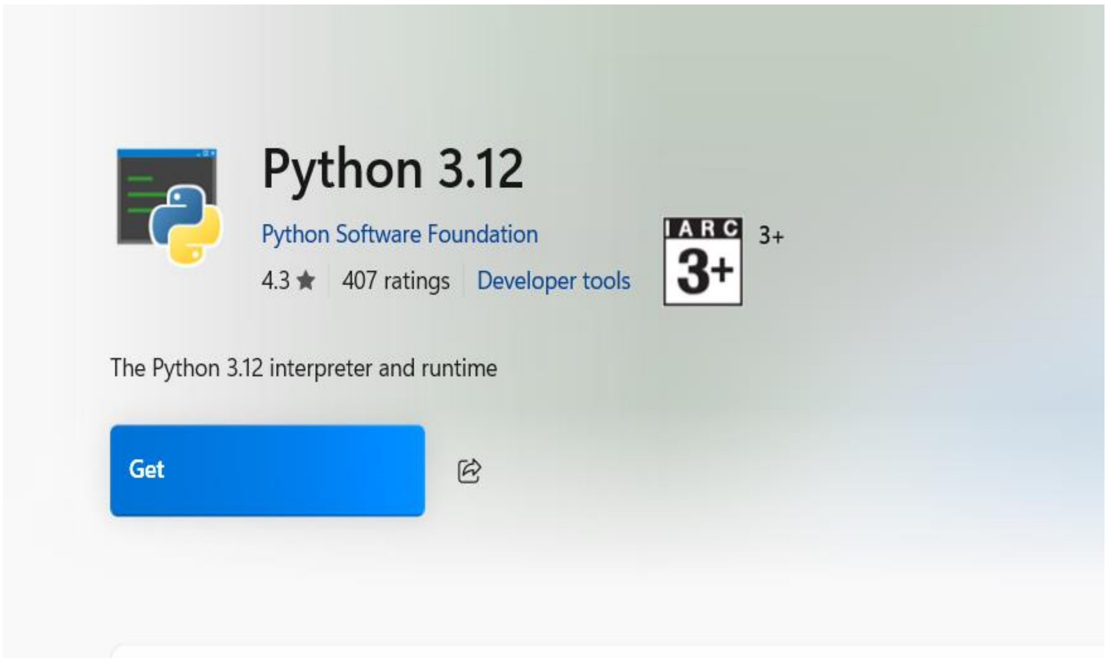
Python Script to Monitor an Application

1. **Proactive Monitoring**: The script ensures continuous monitoring of your application's health, allowing you to detect downtime before it impacts users.
2. **Real-Time Alerts**: Sending instant email notifications enables quick action, reducing the response time in addressing application issues.
3. **Improved Reliability**: Automated monitoring helps maintain application uptime, ensuring a more reliable service for users.
4. **Cost Efficiency**: By identifying and fixing issues early, you can avoid costly downtime and potential revenue loss.
5. **Skill Enhancement**: Writing and implementing this script improves your skills in web monitoring, error handling, and email automation.
6. **Scalable Monitoring**: This PoC can be expanded to monitor multiple applications or integrated into a larger system for enterprise-level monitoring

Step-by-Step Overview

Step 1: Install Python from Microsoft Store

1. Open the **Microsoft Store** on your computer.
2. In the search bar, type **"Python"** and press **Enter**.
3. Find the latest version of Python (e.g., **Python 3.x.x**), and click on it.
4. Click the **Install** button to install Python on your system.
   - This will automatically add Python to your system's PATH environment variable.

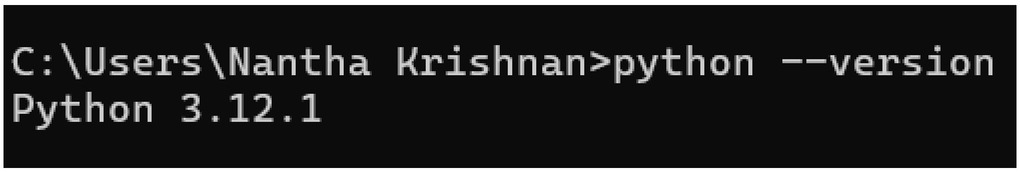

Python 3.12
Python Software Foundation
4.3 ★
407 ratings
Developer tools
IARC 3+
3+
The Python 3.12 interpreter and runtime
Get

Step 2: Verify Python Installation

1. Open the **Command Prompt (CMD)**:
2. Type the following command to verify that Python is installed:

```
python --version
```

3. This should return the version of Python installed, e.g., `Python 3.x.x`.
4. If you see the version number, Python is correctly installed.



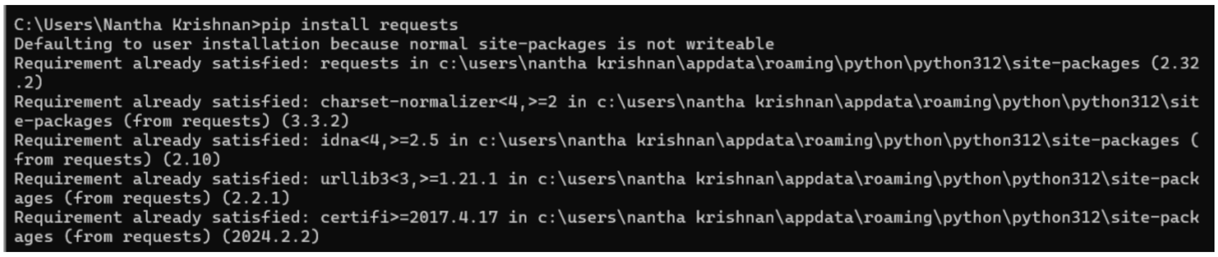
```
C:\Users\Nantha Krishnan>python --version
Python 3.12.1
```

Step 3: Install Required Libraries (requests, smtplib)

1. In **Command Prompt (CMD)**, type the following command to install the **requests** library:

```
pip install requests
```

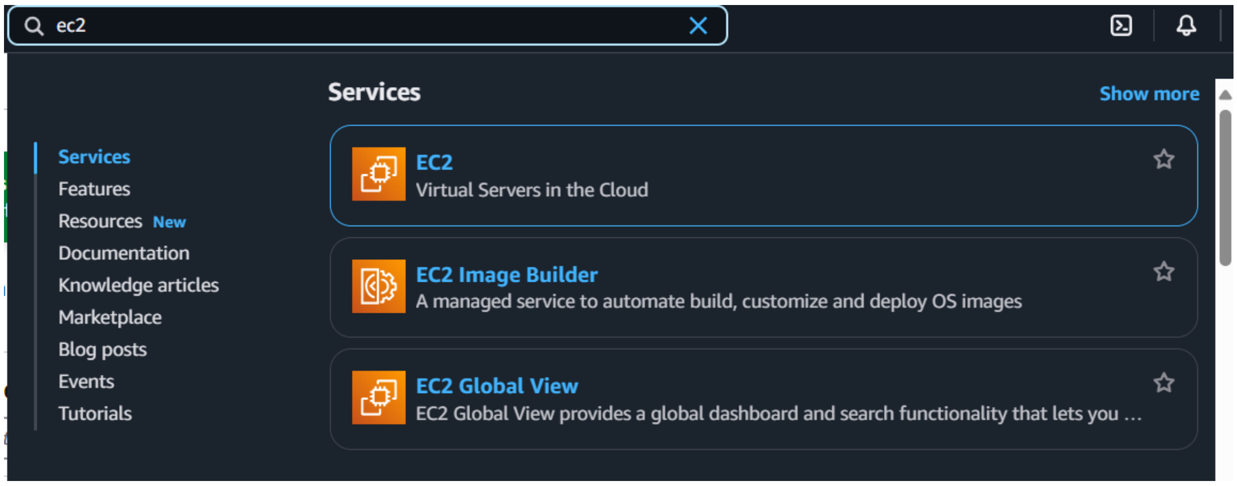
2. The **smtplib** library is included with Python by default, so no installation is needed for it.



```
C:\Users\Nantha Krishnan>pip install requests
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: requests in c:\users\nantha krishnan\appdata\roaming\python\python312\site-packages (2.32
.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\nantha krishnan\appdata\roaming\python\python312\sit
e-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\nantha krishnan\appdata\roaming\python\python312\site-packages (
from requests) (2.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\nantha krishnan\appdata\roaming\python\python312\site-pack
ages (from requests) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\nantha krishnan\appdata\roaming\python\python312\site-pack
ages (from requests) (2024.2.2)
```

Step 4: Write the Python Script

1. Create a EC2 Instance
2. Open any **text editor** (e.g., Notepad, VS Code).
3. Copy and paste the Python script to monitor your EC2 instance (from your PoC).
4. Change your_email@example.com to your actual Gmail address (e.g., your_email@gmail.com).
5. Set smtp_user to your **Gmail** address as well.
6. Enter your **app-specific password** (not your Gmail password) for the smtp_password field. If you don't have an app-specific password, you can create one in your Google Account settings (in the **Security** section under **App passwords**)
7. Also Change the app_url to your Instance URL
8. Save the file with a **.py** extension, e.g., monitor_app.py

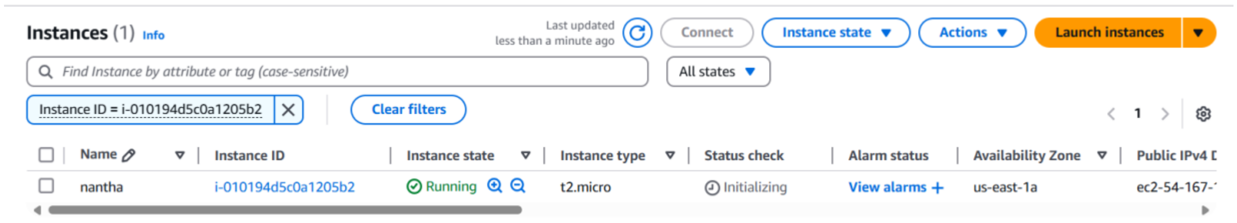


Services

EC2
Virtual Servers in the Cloud

EC2 Image Builder
A managed service to automate build, customize and deploy OS images

EC2 Global View
EC2 Global View provides a global dashboard and search functionality that lets you …

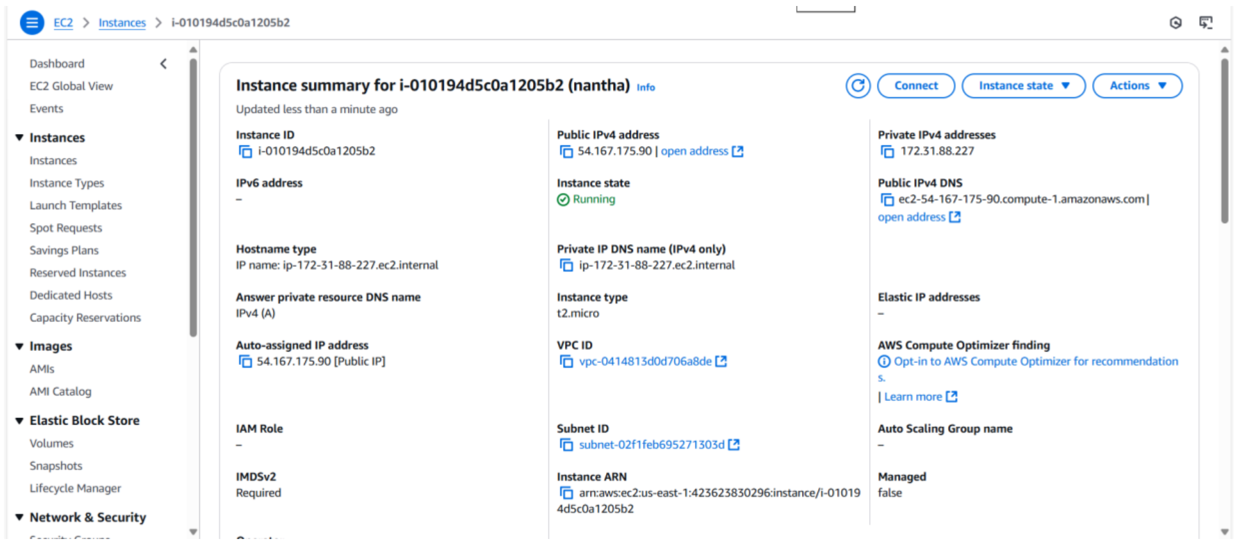


Instances (1) Info

Find Instance by attribute or tag (case-sensitive)

Instance ID = i-010194d5c0a1205b2

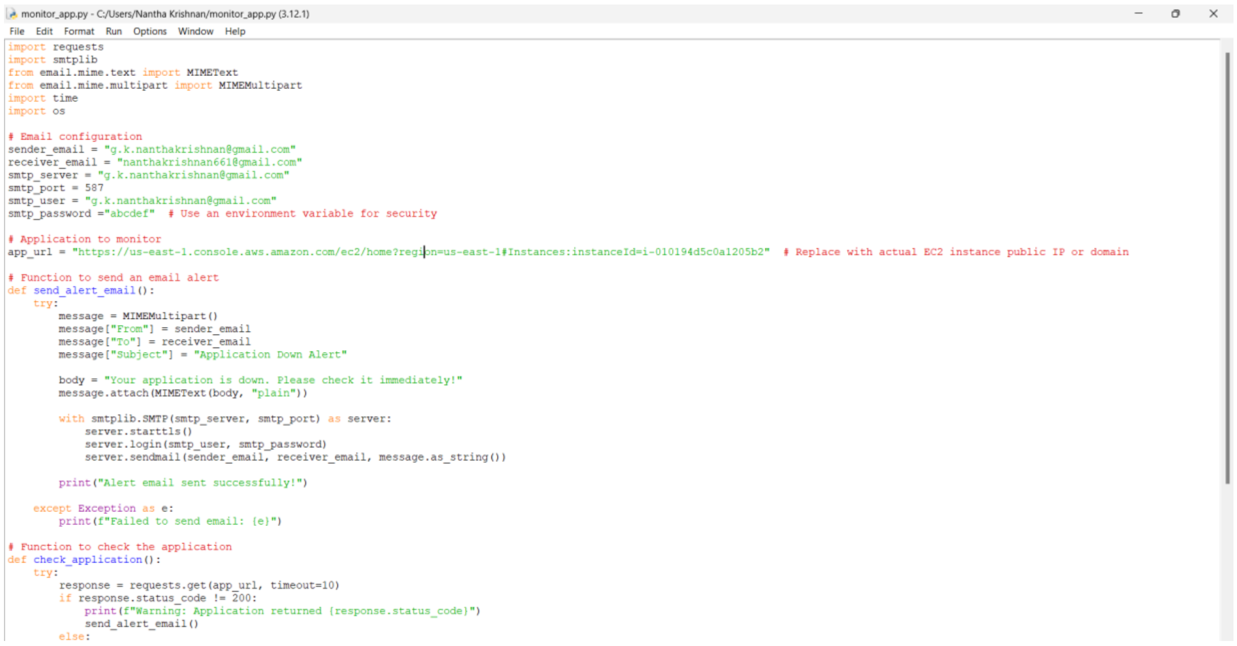
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 D
nantha	i-010194d5c0a1205b2	Running	t2.micro	Initializing	View alarms +	us-east-1a	ec2-54-167-



Instance summary for i-010194d5c0a1205b2 (nantha) Info

Updated less than a minute ago

Instance ID i-010194d5c0a1205b2	Public IPv4 address 54.167.175.90 | open address	Private IPv4 addresses 172.31.88.227
IPv6 address –	Instance state Running	Public IPv4 DNS ec2-54-167-175-90.compute-1.amazonaws.com | open address
Hostname type IP name: ip-172-31-88-227.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-88-227.ec2.internal	
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	Elastic IP addresses –
Auto-assigned IP address 54.167.175.90 [Public IP]	VPC ID vpc-0414813d0d706a8de	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role –	Subnet ID subnet-02f1feb695271303d	Auto Scaling Group name –
IMDSv2 Required	Instance ARN arn:aws:ec2:us-east-1:423623830296:instance/i-010194d5c0a1205b2	Managed false

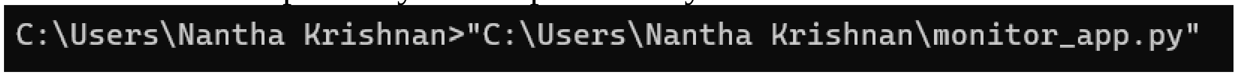

monitor_app.py - C:/Users/Nantha Krishnan/monitor_app.py (3.12.1)
File Edit Format Run Options Window Help

```python
import requests
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import time
import os

# Email configuration
sender_email = "g.k.nanthakrishnan@gmail.com"
receiver_email = "nanthakrishnan661@gmail.com"
smtp_server = "g.k.nanthakrishnan@gmail.com"
smtp_port = 587
smtp_user = "g.k.nanthakrishnan@gmail.com"
smtp_password ="abcdef"  # Use an environment variable for security

# Application to monitor
app_url = "https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:instanceId=i-010194d5c0a1205b2"  # Replace with actual EC2 instance public IP or domain

# Function to send an email alert
def send_alert_email():
    try:
        message = MIMEMultipart()
        message["From"] = sender_email
        message["To"] = receiver_email
        message["Subject"] = "Application Down Alert"

        body = "Your application is down. Please check it immediately!"
        message.attach(MIMEText(body, "plain"))

        with smtplib.SMTP(smtp_server, smtp_port) as server:
            server.starttls()
            server.login(smtp_user, smtp_password)
            server.sendmail(sender_email, receiver_email, message.as_string())

        print("Alert email sent successfully!")

    except Exception as e:
        print(f"Failed to send email: {e}")

# Function to check the application
def check_application():
    try:
        response = requests.get(app_url, timeout=10)
        if response.status_code != 200:
            print(f"Warning: Application returned {response.status_code}")
            send_alert_email()
        else:
```

Step 6: Run the Python Script

1. In **Command Prompt (CMD)**, navigate to the folder where the Python script is saved using the cd command:

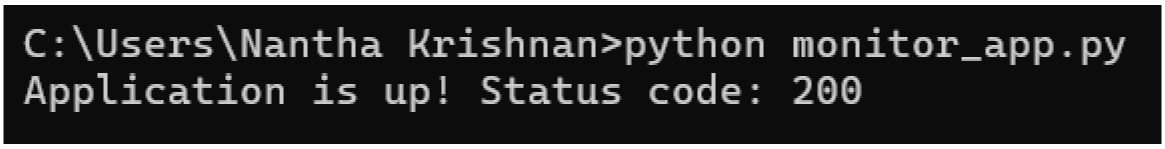
cd path\to\your\script\directory



```
C:\Users\Nantha Krishnan>"C:\Users\Nantha Krishnan\monitor_app.py"
```

2. Run the script with the following command:

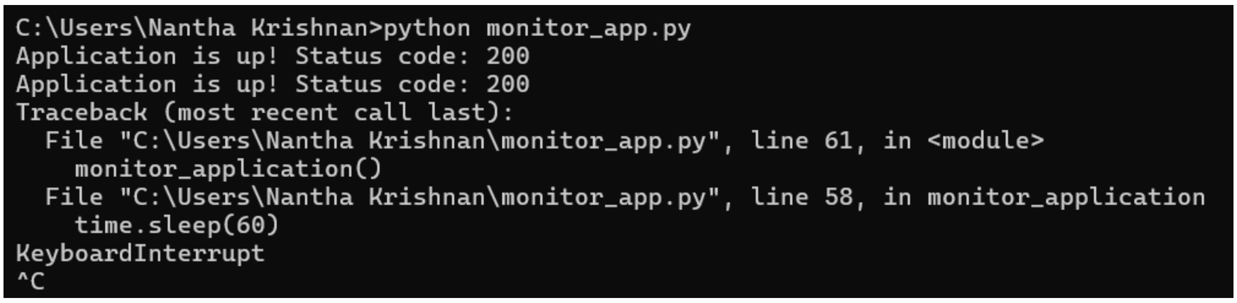
python monitor_app.py



```
C:\Users\Nantha Krishnan>python monitor_app.py
Application is up! Status code: 200
```

Step 7: Stop the Script

To stop the script at any time, press **Ctrl + C** in the **Command Prompt** window.



```
C:\Users\Nantha Krishnan>python monitor_app.py
Application is up! Status code: 200
Application is up! Status code: 200
Traceback (most recent call last):
  File "C:\Users\Nantha Krishnan\monitor_app.py", line 61, in <module>
    monitor_application()
  File "C:\Users\Nantha Krishnan\monitor_app.py", line 58, in monitor_application
    time.sleep(60)
KeyboardInterrupt
^C
```

Outcome

- Monitor Web Application Health**: Periodically send HTTP requests to your application to verify if it is up and running.**

- **Automated Alerts**: Automatically send email alerts whenever the application is down or unreachable, ensuring quick response time.

- **Error Handling**: Implement error handling to detect and respond to network issues, timeout errors, and non-200 HTTP responses.

- **Script Automation**: Run the script in an automated manner (every 60 seconds or as configured) to continuously monitor the application's availability.

- **Reliability and Maintenance**: Improve application reliability by ensuring it's monitored in real-time and receive alerts on downtime or issues that need attention.

- **Email Notification System**: Implement an email notification system using **SMTP** to ensure that administrators or relevant personnel are promptly informed of application downtime.