PW SKILLS | DevOps and Cloud Computing

Automating System Administration Tasks with Python

Objective

- Automate system administration tasks such as backups, log parsing, and notifications.
- Use Python libraries (shutil, os, logging, smtplib) to manage system operations.
- Combine multiple Python libraries to build robust DevOps automation scripts.
- Develop a complete automation workflow for a real-world system task.

PW SKILLS

The importance of automated backups for system reliability

Importance of automated backup

Automated backups are critical for ensuring system reliability and data integrity in today's digital landscape.

Here are the key reasons why automated backups are essential for system reliability:

1. Enhanced Data Security
2. Reduction of Human Error
3. Time Efficiency and Resource Optimization
4. Improved Recovery Point Objectives (RPO) and Recovery Time Objectives (RTO)
5. Scalability and Flexibility
6. Compliance and Reporting

PW SKILLS

Demonstrating file and directory operations using shutil (copy, move, make archive)

Let’s do it

1. **Copying a File**

To copy a file using shutil, follow these steps:

- Create a sample file.
- Use ‘shutil.copy’ to create a copy of that file.

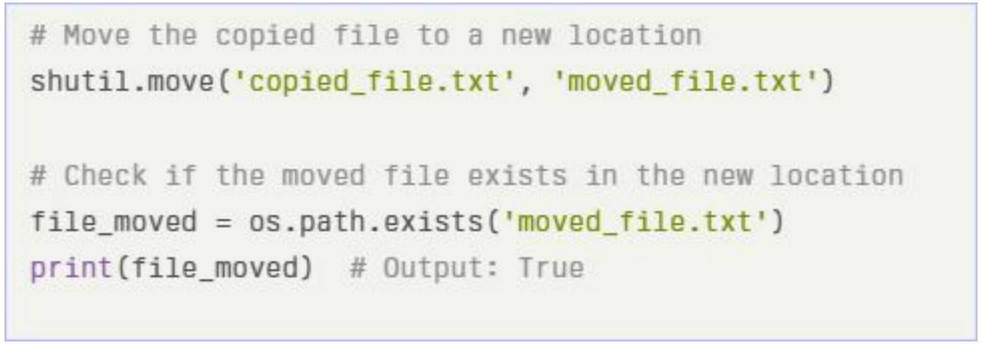
```
import shutil
import os

# Create a sample file to copy
with open('sample_file.txt', 'w') as f:
    f.write('This is a sample file for shutil operations.')

# Copy the file to a new location
shutil.copy('sample_file.txt', 'copied_file.txt')

# Check if the copied file exists
file_copied = os.path.exists('copied_file.txt')
print(file_copied)  # Output: True
```

Let’s do it

2. Moving a File

To move the copied file to a new location:

- Use ‘shutil.move’ to relocate the file.



```python
# Move the copied file to a new location
shutil.move('copied_file.txt', 'moved_file.txt')

# Check if the moved file exists in the new location
file_moved = os.path.exists('moved_file.txt')
print(file_moved)  # Output: True
```

Let’s do it

3. Creating an Archive of a Directory

To create an archive of a directory using shutil:

- Create a sample directory with files.
- Use ‘shutil.make-archive’ to create a zip archive of that directory.

```
# Create a sample directory with files
os.makedirs('sample_dir', exist_ok=True)
with open('sample_dir/file1.txt', 'w') as f:
    f.write('File 1 content')
with open('sample_dir/file2.txt', 'w') as f:
    f.write('File 2 content')

# Create an archive of the directory
shutil.make_archive('sample_archive', 'zip', 'sample_dir')

# Check if the archive was created
archive_created = os.path.exists('sample_archive.zip')
print(archive_created)  # Output: True
```

Showing how to automate file backups with timestamped archive creation

Let’s do it

To automate file backups with timestamped archive creation using Python's ‘shutil’ and ‘datetime’ modules, you can follow these steps:

1. Import Required Modules
2. Define Source and Destination
3. Create a Timestamp
4. Make the Archive

Here’s a concise example:

```python
import shutil
import os
from datetime import datetime

# Define the source directory to back up
source_dir = 'path/to/source_directory'

# Define the destination directory for the backup
backup_dir = 'path/to/backup_directory'

# Create a timestamp for the backup archive
timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')

# Create a backup archive with a timestamp
backup_archive_name = os.path.join(backup_dir, f'backup_{timestamp}')
shutil.make_archive(backup_archive_name, 'zip', source_dir)

print(f'Backup created: {backup_archive_name}.zip')
```

Pop Quiz

Q. What is the primary benefit of timestamping backup archives?

A

To differentiate between multiple backups taken at different times

B

To compress backup files

Pop Quiz

Q. What is the primary benefit of timestamping backup archives?

A

To differentiate between multiple backups taken at different times

B

To compress backup files

Explaining log management in system administration

Log management

Log management is a crucial aspect of system administration that involves the systematic collection, storage, analysis, and monitoring of log data generated by various systems and applications.

Importance of Log Management:

- Security Monitoring
- Performance Monitoring
- Centralized Data Access

Log management

Log Management Process:

1. Collection
2. Aggregation
3. Storage
4. Analysis
5. Reporting
6. Action

PW SKILLS

Introducing Python's logging module and how to read and analyze logs

Python’s logging module

Python's ‘logging’ module is a built-in library that provides a flexible and powerful way to record events that occur during the execution of a program. It allows developers to track information, debug code, and monitor applications effectively.

Key Components and Concepts:

- Loggers
- Log Levels
- Handlers
- Formatters

Python’s logging module

Basic Usage:

1. Import the ‘logging’ module:

```
import logging
```

2. Configure Logging:

```
logging.basicConfig(filename='example.log', level=logging.DEBUG,
                    format='%(asctime)s - %(name)s - %(levelname)s -
%(message)s')
```

3. Log Messages:

```
logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

Python’s logging module

Reading and Analyzing Logs:

1. Open the Log File:

```
with open('example.log', 'r') as file:
    for line in file:
        print(line.strip())
```

2. Parse Log Entries: Analyze each line to extract relevant information such as timestamps, log levels, and messages.
3. Use Regular Expressions: Employ regular expressions for more complex filtering and analysis of log entries.

```
import re

with open('example.log', 'r') as file:
    for line in file:
        if re.search(r'ERROR', line):
            print(line.strip())
```

PW SKILLS

Demonstrating searching for keywords or error patterns in system logs using regular expressions (re).

Let’s do it

1. Import the ‘re’ Module:

```python
import re
```

2. Define the Log File Path:

```python
log_file_path = 'path/to/your/logfile.log'
```

3. Read the Log File:

```python
with open(log_file_path, 'r') as file:
    logs = file.readlines()
```

Let’s do it

4. Define Regular Expressions for Keywords or Patterns:

```
error_pattern = r'ERROR'
warning_pattern = r'WARNING'
```

5. 5. Search for Patterns in Logs:

```
for line in logs:
    if re.search(error_pattern, line):
        print(f'Error found: {line.strip()}')
    elif re.search(warning_pattern, line):
        print(f'Warning found: {line.strip()}')
```

PW SKILLS

Take A 5-Minute Break!

- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes

PW SKILLS

The importance of automated alerts in system monitoring

Importance of automated alerts

Automated alerts play a vital role in system monitoring by providing real-time notifications that help organizations respond swiftly to potential issues.

Here are the key reasons highlighting their importance:

1. Immediate Response to Issues
2. Continuous Monitoring
3. Enhanced Accuracy and Reduced Human Error
4. Proactive Issue Detection
5. Improved Operational Efficiency
6. Cost Savings
7. Customizable Notifications

Introducing smtplib and email modules for sending email notifications.

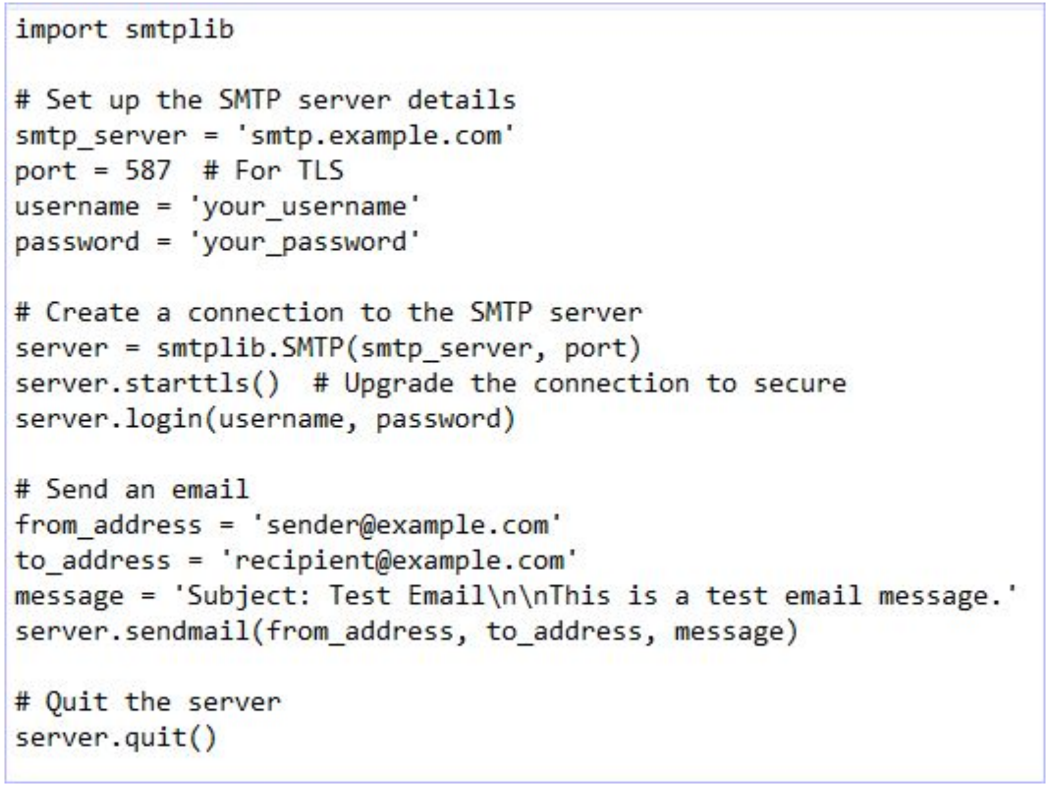
smtplib & email modules

smtplib Module
The smtplib module is part of Python's standard library and provides a simple interface for sending emails using the Simple Mail Transfer Protocol (SMTP). It allows you to connect to an SMTP server, authenticate, and send email messages.

Key Features:

- SMTP Client Session
- Connection Management

Basic Usage Example:



```python
import smtplib

# Set up the SMTP server details
smtp_server = 'smtp.example.com'
port = 587  # For TLS
username = 'your_username'
password = 'your_password'

# Create a connection to the SMTP server
server = smtplib.SMTP(smtp_server, port)
server.starttls()  # Upgrade the connection to secure
server.login(username, password)

# Send an email
from_address = 'sender@example.com'
to_address = 'recipient@example.com'
message = 'Subject: Test Email\n\nThis is a test email message.'
server.sendmail(from_address, to_address, message)

# Quit the server
server.quit()
```

smtplib & email modules

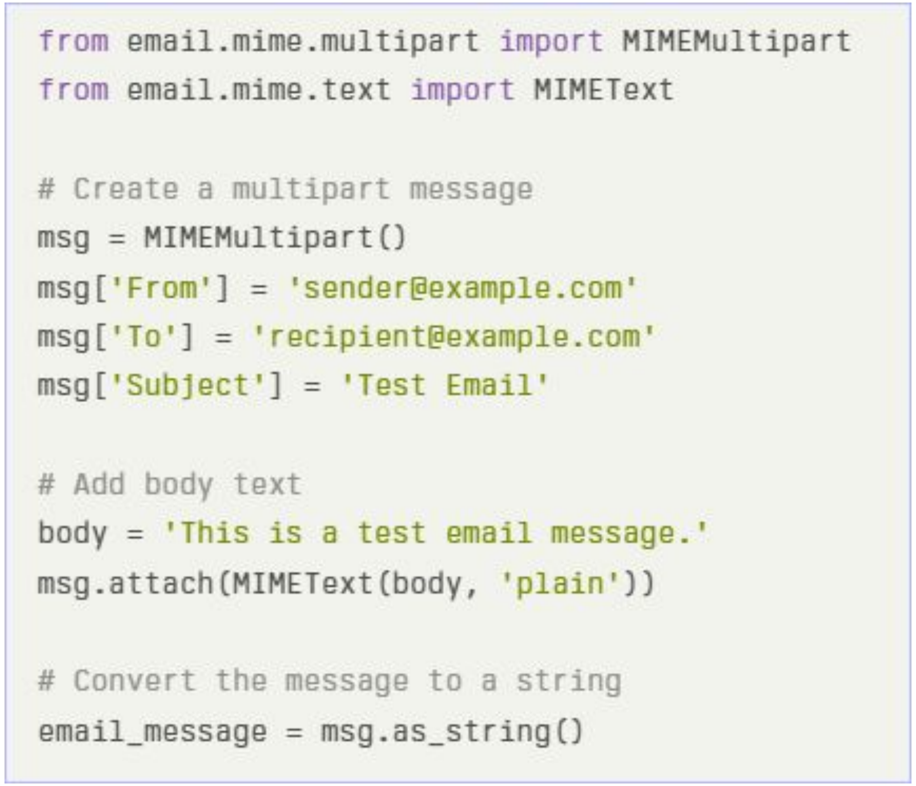
email Module

The email module is also part of Python's standard library and provides tools for constructing and parsing email messages. It allows you to create complex email structures, including plain text, HTML content, and attachments.

Key Features:

- MIME Support
- EmailMessage Class

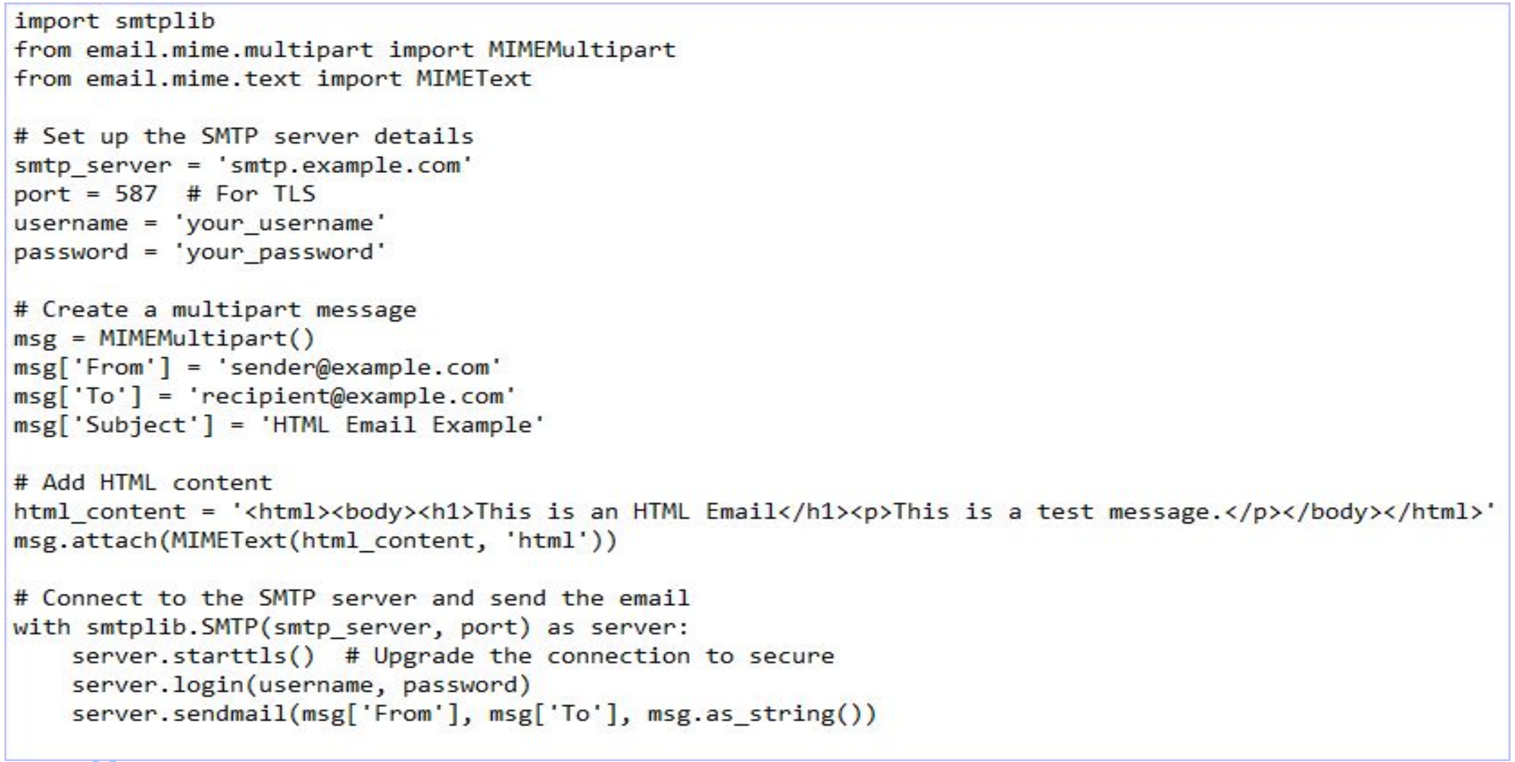
Basic Usage Example:



```python
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

# Create a multipart message
msg = MIMEMultipart()
msg['From'] = 'sender@example.com'
msg['To'] = 'recipient@example.com'
msg['Subject'] = 'Test Email'

# Add body text
body = 'This is a test email message.'
msg.attach(MIMEText(body, 'plain'))

# Convert the message to a string
email_message = msg.as_string()
```

Combining smtplib and email Modules:

To send an email with rich content (like HTML) or attachments, you typically combine both modules. Here’s how you can do that:



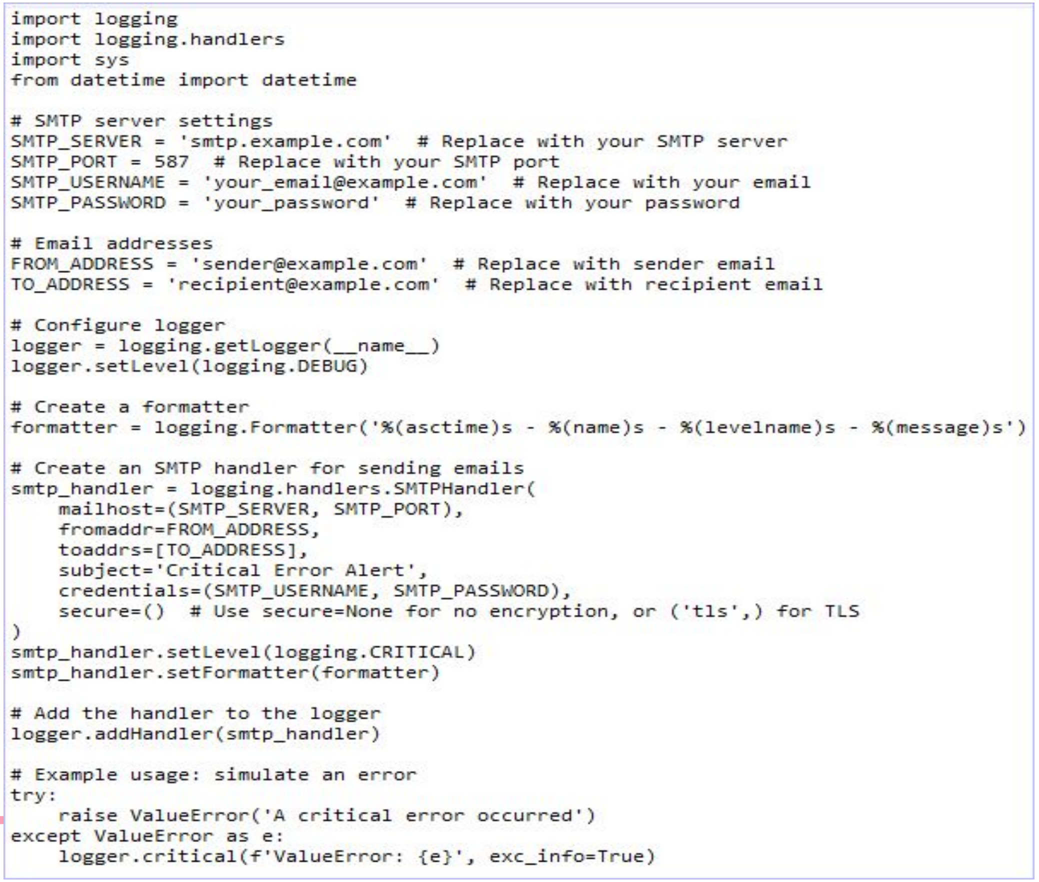
```python
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

# Set up the SMTP server details
smtp_server = 'smtp.example.com'
port = 587  # For TLS
username = 'your_username'
password = 'your_password'

# Create a multipart message
msg = MIMEMultipart()
msg['From'] = 'sender@example.com'
msg['To'] = 'recipient@example.com'
msg['Subject'] = 'HTML Email Example'

# Add HTML content
html_content = '<html><body><h1>This is an HTML Email</h1><p>This is a test message.</p></body></html>'
msg.attach(MIMEText(html_content, 'html'))

# Connect to the SMTP server and send the email
with smtplib.SMTP(smtp_server, port) as server:
    server.starttls()  # Upgrade the connection to secure
    server.login(username, password)
    server.sendmail(msg['From'], msg['To'], msg.as_string())
```

Demonstrating sending an email alert when specific log patterns

Let’s do it

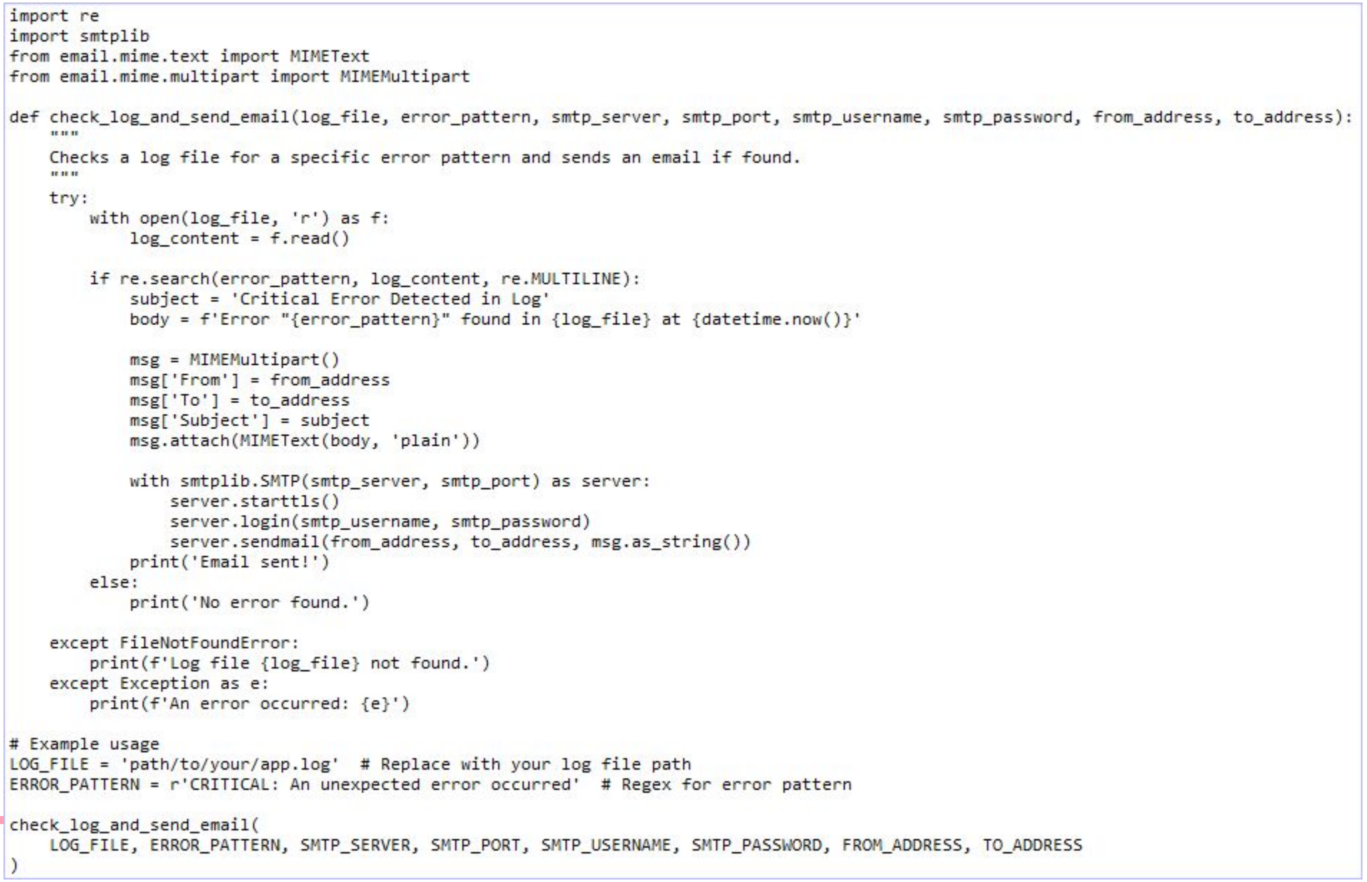
To send an email alert when specific log patterns (e.g., critical errors) are detected, you can use Python's ‘logging’ module along with ‘smtplib’ and ‘email’. Here's a demonstration:

1. Configure Logging to Send Email Alerts:



```python
import logging
import logging.handlers
import sys
from datetime import datetime

# SMTP server settings
SMTP_SERVER = 'smtp.example.com'  # Replace with your SMTP server
SMTP_PORT = 587  # Replace with your SMTP port
SMTP_USERNAME = 'your_email@example.com'  # Replace with your email
SMTP_PASSWORD = 'your_password'  # Replace with your password

# Email addresses
FROM_ADDRESS = 'sender@example.com'  # Replace with sender email
TO_ADDRESS = 'recipient@example.com'  # Replace with recipient email

# Configure logger
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

# Create a formatter
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

# Create an SMTP handler for sending emails
smtp_handler = logging.handlers.SMTPHandler(
    mailhost=(SMTP_SERVER, SMTP_PORT),
    fromaddr=FROM_ADDRESS,
    toaddrs=[TO_ADDRESS],
    subject='Critical Error Alert',
    credentials=(SMTP_USERNAME, SMTP_PASSWORD),
    secure=()  # Use secure=None for no encryption, or ('tls',) for TLS
)
smtp_handler.setLevel(logging.CRITICAL)
smtp_handler.setFormatter(formatter)

# Add the handler to the logger
logger.addHandler(smtp_handler)

# Example usage: simulate an error
try:
    raise ValueError('A critical error occurred')
except ValueError as e:
    logger.critical(f'ValueError: {e}', exc_info=True)
```

2. Create a Function to Monitor Logs and Send Alerts:



```
import re
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

def check_log_and_send_email(log_file, error_pattern, smtp_server, smtp_port, smtp_username, smtp_password, from_address, to_address):
    """
    Checks a log file for a specific error pattern and sends an email if found.
    """
    try:
        with open(log_file, 'r') as f:
            log_content = f.read()

        if re.search(error_pattern, log_content, re.MULTILINE):
            subject = 'Critical Error Detected in Log'
            body = f'Error "{error_pattern}" found in {log_file} at {datetime.now()}'

            msg = MIMEMultipart()
            msg['From'] = from_address
            msg['To'] = to_address
            msg['Subject'] = subject
            msg.attach(MIMEText(body, 'plain'))

            with smtplib.SMTP(smtp_server, smtp_port) as server:
                server.starttls()
                server.login(smtp_username, smtp_password)
                server.sendmail(from_address, to_address, msg.as_string())
            print('Email sent!')
        else:
            print('No error found.')

    except FileNotFoundError:
        print(f'Log file {log_file} not found.')
    except Exception as e:
        print(f'An error occurred: {e}')

# Example usage
LOG_FILE = 'path/to/your/app.log'  # Replace with your log file path
ERROR_PATTERN = r'CRITICAL: An unexpected error occurred'  # Regex for error pattern

check_log_and_send_email(
    LOG_FILE, ERROR_PATTERN, SMTP_SERVER, SMTP_PORT, SMTP_USERNAME, SMTP_PASSWORD, FROM_ADDRESS, TO_ADDRESS
)
```

PW SKILLS

Discussing real-world DevOps automation workflows

Let’s discuss

DevOps automation workflows streamline software development and operations by automating repetitive tasks.

Key components of DevOps automation workflows:

1. Continuous Integration/Continuous Deployment (CI/CD)
   a. Automated Testing
   b. Deployment Automation
2. Infrastructure as Code (IaC)
3. Monitoring and Incident Response

Let’s discuss

Benefits of Automation in DevOps Workflows:

- Speed
- Consistency
- Improved collaboration
- Enhanced security

PW SKILLS

Demonstrating combining multiple Python libraries

Let’s do it

Here’s a demonstration of combining multiple Python libraries for various automation tasks:

```
import shutil
import os
import subprocess
import re
import smtplib
from email.mime.text import MIMEText

# Step 1: Backup files using shutil
def backup files(source dir, backup dir):
    if not os.path.exists(backup dir):
        os.makedirs(backup dir)
    shutil.copytree(source dir, backup dir)
    print(f"Backup completed from {source dir} to {backup dir}")

# Step 2: Execute a system command using subprocess
def run command(command):
    result = subprocess.run(command, shell=True, capture output=True, text=True)
    print(f"Command output: {result.stdout}")
    return result
```

```
# Step 3: Parse logs using re
def parse logs(log file):
    with open(log file, 'r') as file:
        logs = file.readlines()
    error lines = [line for line in logs if re.search(r'ERROR', line)]
    return error lines

# Step 4: Send notification email using smtplib
  def send notification(subject, body, recipient email):
    sender email = "your email@example.com"
    msg = MIMEText(body)
    msg['Subject'] = subject
    msg['From'] = sender email
    msg['To'] = recipient email

    with smtplib.SMTP('smtp.example.com', 587) as server:
        server.starttls()
        server.login(sender email, 'your password')
        server.sendmail(sender email, recipient email, msg.as string())
        print("Notification sent.")
```

```
# Example usage
if __name__ == "__main__":
    # Backup operation
    backup_files('/path/to/source', '/path/to/backup')

    # Run a system command
    run command('ls -la')

    # Parse log file for errors
    errors = parse logs('/path/to/logfile.log')
    if errors:
        send notification("Log Errors Detected", "\n".join(errors), "recipient@example.com")
```

PW SKILLS

Time for case study!

Objective

- Complete the post-class assessment
- Complete assignments (if any)
- Practice the concepts and techniques taught in this session
- Review your lecture notes
- Note down questions and queries regarding this session and consult the teaching assistants

Thanks
!