Common DevOps Automation Use Cases with Python: - Yatri Cloud

Creator: Yatharth Chauhan

Automation of repetitive tasks is a fundamental practice in DevOps, which involves automating the deployment, management, and monitoring of infrastructure and applications. Python is a widely-used language for DevOps automation due to its simplicity and the vast ecosystem of libraries.

1. **Infrastructure Provisioning** (e.g., using Terraform or AWS SDK)
2. **Configuration Management** (e.g., automating server setup or updating configurations)
3. **Continuous Integration/Continuous Deployment (CI/CD) Pipelines** (e.g., integrating with Jenkins)
4. **Log File Monitoring and Alerting** (e.g., tailing logs, sending notifications)
5. **Backup and Restore Tasks** (e.g., automating database backups)

Example 1: Automating Server Setup

We can use Python scripts to interact with cloud providers like AWS to automate the provisioning of servers (EC2 instances). Using AWS's SDK for Python (`boto3`), we can automate the creation and management of cloud infrastructure.

```python
import boto3
from botocore.exceptions import NoCredentialsError, ClientError

# Initialize a session using Amazon EC2
ec2 = boto3.resource('ec2')

# Function to launch a new EC2 instance
def launch_ec2_instance():
    try:
        # Launch an EC2 instance
        instance = ec2.create_instances(
            ImageId='ami-0c55b159cbfafe1f0',  # Amazon Linux 2 AMI ID (replace
with valid one)
            MinCount=1,
            MaxCount=1,
            InstanceType='t2.micro',  # Free tier instance type
            KeyName='yatricloud-key-pair'     # SSH Key for access
        )
        print(f'Instance launched: {instance[0].id}')
    except NoCredentialsError:
        print("Credentials not available")
    except ClientError as e:
        print(f"Error occurred: {e}")

launch_ec2_instance()
```

Explanation:

- `boto3`: This is AWS's SDK for Python, allowing you to programmatically interact with AWS services.

- **Key elements**:
  - `ec2.create_instances()`: This function automates the process of launching new EC2 instances.
  - `ImageId`: The Amazon Machine Image (AMI) you want to use for the instance.
  - `InstanceType`: Specifies the type of the instance (e.g., `t2.micro` for a free-tier eligible instance).
  - `KeyName`: This is used to specify an SSH key pair for secure access to the instance.

Example 2: Automating Configuration Management using Ansible

In DevOps, tools like **Ansible** are used to manage configuration files across multiple servers. Python can be used to automate Ansible playbooks.

```python
import os

# Automating an Ansible playbook using Python
def run_ansible_playbook(playbook_path):
    try:
        os.system(f"ansible-playbook {playbook_path}")
        print("Playbook executed successfully")
    except Exception as e:
        print(f"Error while executing playbook: {e}")

run_ansible_playbook('/path/to/playbook.yml')
```

Explanation:

- `os.system()`: Executes a system command. Here, it's used to run an Ansible playbook from a Python script.
- This allows you to automate repetitive configuration management tasks, such as setting up application servers or updating configurations on multiple servers.

Example 3: Automating CI/CD Pipeline with Jenkins API

Python can be used to trigger Jenkins jobs and manage CI/CD pipelines programmatically.

```python
import requests
from requests.auth import HTTPBasicAuth

# Trigger Jenkins job using Jenkins API
def trigger_jenkins_job(job_name, jenkins_url, username, token):
    try:
        # Construct the URL for triggering the job
        job_url = f'{jenkins_url}/job/{job_name}/build'

        # Trigger the Jenkins job
        response = requests.post(job_url, auth=HTTPBasicAuth(username, token))

        if response.status_code == 201:
            print(f"Job '{job_name}' triggered successfully")
        else:
            print(f"Failed to trigger job. Status code: {response.status_code}")
    except Exception as e:
        print(f"Error occurred: {e}")

trigger_jenkins_job('yatricloud-job', 'http://yatricloud-jenkins-server',
'yatharth-chauhan', 'yatricloud-token')
```

Explanation:

- `requests`: This library allows us to make HTTP requests to the Jenkins API to trigger a build.
- `HTTPBasicAuth()`: Used for authentication with Jenkins, passing the username and API token.
- **Jenkins API**: Jenkins provides a REST API that can be used to trigger builds, check job statuses, etc.

Example 4: Automating Log File Monitoring and Alerts

Python can be used to monitor logs and trigger alerts when specific patterns or errors appear in log files.

```
import time

# Function to monitor a log file
def monitor_log_file(log_file_path, alert_string):
    with open(log_file_path, 'r') as file:
        file.seek(0, 2)  # Move the cursor to the end of the file
        while True:
            line = file.readline()
            if not line:
                time.sleep(1)  # Sleep briefly and wait for new lines
                continue
            if alert_string in line:
                print(f"Alert! Found string '{alert_string}' in log file")
                # Additional alerting logic can go here (e.g., send an email,
Slack message)

# Example usage
monitor_log_file('/var/log/yatricloud.log', 'ERROR')
```

Explanation:

- **Log file monitoring**: This script opens the log file and continuously reads it to check for specific strings (e.g., `ERROR`).
- When a match is found, an alert message is printed, and additional alerting mechanisms (like sending an email) can be implemented.

Benefits of Automating Repetitive Tasks with Python:

1. **Efficiency**: Automation saves time and effort by performing repetitive tasks without human intervention.
2. **Consistency**: Ensures tasks are executed the same way every time, reducing the chances of errors.

3. **Scalability**: Automated processes can easily scale, whether deploying 1 or 100 servers.
4. **Cost-Saving**: By reducing manual work, companies save resources that can be utilized elsewhere.

In DevOps, automation is essential to improve the speed, efficiency, and reliability of operations. Python, with its simplicity and rich set of libraries, provides an excellent choice for automating various DevOps tasks—from infrastructure provisioning to CI/CD pipelines and log monitoring. The examples above provide a starting point for automating repetitive tasks in your workflow.
