

# Automation with Shell Scripting & Python in DevOps

In DevOps, automation is key for improving efficiency, reducing errors, and speeding up deployments. Python and Shell scripting are both widely used, each offering unique advantages depending on the task.

## When to Use Shell Scripting:

- **System Administration:** Automate tasks like file management, user permissions, and service control.
  - **Command Line Interactions:** Ideal for interacting with command-line tools (e.g., for logging and container management).
  - **Infrastructure Automation:** Used in tools like Ansible for server provisioning and maintenance.
  - **Quick Prototyping:** Fast to write for simple, one-off tasks.
  - **Log and Text Processing:** Efficient for parsing and extracting data from logs.
- 

## When to Use Python:

- **Complex Workflows:** Ideal for multi-step automation, APIs, and handling intricate logic.
- **Cross-Platform:** Works seamlessly across Linux, Windows, and macOS.
- **CI/CD Pipeline Automation:** Integrates well with tools like Jenkins and GitLab CI.
- **API and Cloud Integration:** Excellent for interacting with cloud services and infrastructure tools.
- **Scalable and Reusable Code:** Python is great for large automation projects requiring maintainable code.

## Key Differences:

- **Shell Scripting:** Best for system-level tasks and rapid automation; tied to Unix-like environments.
- **Python:** More versatile, with a vast ecosystem of libraries, making it suitable for complex, cross-platform automation.

## Conclusion:

Shell scripting is great for quick, system-level tasks, while Python excels in complex, scalable automation. Using both together can create a flexible and efficient DevOps automation strategy.

---

# Automation with Shell scripting for DevOps

## 1. Automating Server Provisioning (AWS EC2 Launch)

```
#!/bin/bash
```

### # Variables

```
INSTANCE_TYPE="t2.micro"
```

```
AMI_ID="ami-0abcdef1234567890" # Replace with the correct AMI ID
```

```
KEY_NAME="my-key-pair" # Replace with your key pair name
```

```
SECURITY_GROUP="sg-0abc1234def567890" # Replace with your security group ID
```

```
SUBNET_ID="subnet-0abc1234def567890" # Replace with your subnet ID
```

```
REGION="us-west-2" # Replace with your AWS region
```

### **# Launch EC2 instance**

```
aws ec2 run-instances --image-id $AMI_ID --count 1 --instance-type  
$INSTANCE_TYPE \
```

```
--key-name $KEY_NAME --security-group-ids $SECURITY_GROUP --subnet-id  
$SUBNET_ID --region $REGION
```

```
echo "EC2 instance launched successfully!"
```

---

## **2. System Monitoring (CPU Usage Alert)**

```
#!/bin/bash
```

### **# Threshold for CPU usage**

```
CPU_THRESHOLD=80
```

### **# Get the current CPU usage**

```
CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | sed "s/.*, *\[0-9.\]*\)%* id.*^1/" | awk  
'{print 100 - $1}')
```

**# Check if CPU usage exceeds threshold**

```
if (( $(echo "$CPU_USAGE > $CPU_THRESHOLD" | bc -l) )); then
```

```
    echo "Alert: CPU usage is above $CPU_THRESHOLD%. Current usage is  
$CPU_USAGE%" | mail -s "CPU Usage Alert" user@example.com
```

```
fi
```

---

### **3. Backup Automation (MySQL Backup)**

```
#!/bin/bash
```

**# Variables**

```
DB_USER="root"
```

```
DB_PASSWORD="password"
```

```
DB_NAME="my_database"
```

```
BACKUP_DIR="/backup"
```

```
DATE=$(date +%F)
```

**# Create backup directory if it doesn't exist**

```
mkdir -p $BACKUP_DIR
```

### **# Backup command**

```
mysqldump -u $DB_USER -p$DB_PASSWORD $DB_NAME >  
$BACKUP_DIR/backup_$(date +%Y%m%d).sql
```

### **# Optional: Compress the backup**

```
gzip $BACKUP_DIR/backup_$(date +%Y%m%d).sql
```

```
echo "Backup completed successfully!"
```

---

## **4. Log Rotation and Cleanup**

```
#!/bin/bash
```

### **# Variables**

```
LOG_DIR="/var/log/myapp"
```

```
ARCHIVE_DIR="/var/log/myapp/archive"
```

```
DAYS_TO_KEEP=30
```

### **# Create archive directory if it doesn't exist**

```
mkdir -p $ARCHIVE_DIR
```

### **# Find and compress logs older than 7 days**

```
find $LOG_DIR -type f -name "*.log" -mtime +7 -exec gzip {} \; -exec mv {}  
$ARCHIVE_DIR \;
```

### **# Delete logs older than 30 days**

```
find $ARCHIVE_DIR -type f -name "*.log.gz" -mtime +$DAYS_TO_KEEP -exec  
rm {} \;
```

```
echo "Log rotation and cleanup completed!"
```

---

## **5. CI/CD Pipeline Automation (Trigger Jenkins Job)**

```
#!/bin/bash
```

### **# Jenkins details**

```
JENKINS_URL="http://jenkins.example.com"
```

```
JOB_NAME="my-pipeline-job"
```

```
USER="your-username"
```

```
API_TOKEN="your-api-token"
```

### **# Trigger Jenkins job**

```
curl -X POST "$JENKINS_URL/job/$JOB_NAME/build" --user  
"$USER:$API_TOKEN"
```

```
echo "Jenkins job triggered successfully!"
```

---

## 6. Deployment Automation (Kubernetes Deployment)

```
#!/bin/bash
```

### # Variables

```
NAMESPACE="default"
```

```
DEPLOYMENT_NAME="my-app"
```

```
IMAGE="my-app:v1.0"
```

### # Deploy to Kubernetes

```
kubectl set image deployment/$DEPLOYMENT_NAME
```

```
$DEPLOYMENT_NAME=$IMAGE --namespace=$NAMESPACE
```

```
echo "Deployment updated to version $IMAGE!"
```

---

## 7. Infrastructure as Code (Terraform Apply)

```
#!/bin/bash
```

```
# Variables
```

```
TF_DIR="/path/to/terraform/config"
```

```
# Navigate to Terraform directory
```

```
cd $TF_DIR
```

```
# Run terraform apply
```

```
terraform apply -auto-approve
```

```
echo "Terraform apply completed successfully!"
```

---

## **8. Database Management (PostgreSQL Schema Migration)**

```
bash
```

```
#!/bin/bash
```

```
# Variables
```



```
DB_USER="postgres"
```

```
DB_PASSWORD="password"
```

```
DB_NAME="my_database"
```

```
MIGRATION_FILE="/path/to/migration.sql"
```

### **# Run schema migration**

```
PGPASSWORD=$DB_PASSWORD psql -U $DB_USER -d $DB_NAME -f  
$MIGRATION_FILE
```

```
echo "Database schema migration completed!"
```

---

## **9. User Management (Add User to Group)**

```
#!/bin/bash
```

### **# Variables**

```
USER_NAME="newuser"
```

```
GROUP_NAME="devops"
```

### **# Add user to group**

```
usermod -aG $GROUP_NAME $USER_NAME
```

```
echo "User $USER_NAME added to group $GROUP_NAME!"
```

---

## 10. Security Audits (Check for Open Ports)

```
#!/bin/bash
```

```
# Check for open ports
```

```
OPEN_PORTS=$(netstat -tuln)
```

```
# Check if any ports are open (excluding localhost)
```

```
if [[ $OPEN_PORTS =~ "0.0.0.0" || $OPEN_PORTS =~ "127.0.0.1" ]]; then
```

```
    echo "Security Alert: Open ports detected!"
```

```
    echo "$OPEN_PORTS" | mail -s "Open Ports Security Alert" user@example.com
```

```
else
```

```
    echo "No open ports detected."
```

```
Fi
```

---

## 11. Performance Tuning

This script clears memory caches and restarts services to free up system resources.

```
#!/bin/bash
```

```
# Clear memory caches to free up resources
```

```
sync; echo 3 > /proc/sys/vm/drop_caches
```

```
# Restart services to free up resources
```

```
systemctl restart nginx
```

```
systemctl restart apache2
```

---

## 12. Automated Testing

This script runs automated tests using a testing framework like pytest for Python or JUnit for Java.

```
#!/bin/bash
```

```
# Run unit tests using pytest (Python example)
```

```
pytest tests/
```

```
# Or, run JUnit tests (Java example)
```

```
mvn test
```

---

## 13. Scaling Infrastructure

This script automatically scales EC2 instances in an Auto Scaling group based on CPU usage.

```
#!/bin/bash
```

```
# Check CPU usage and scale EC2 instances
```

```
CPU_USAGE=$(aws cloudwatch get-metric-statistics --namespace AWS/EC2  
--metric-name CPUUtilization --dimensions  
Name=InstanceId,Value=i-1234567890abcdef0 --statistics Average --period 300  
--start-time $(date -d '5 minutes ago' --utc +%FT%TZ) --end-time $(date --utc  
+%FT%TZ) --query 'Datapoints[0].Average' --output text)
```

```
if (( $(echo "$CPU_USAGE > 80" | bc -l) )); then
```

```
    aws autoscaling update-auto-scaling-group --auto-scaling-group-name  
my-auto-scaling-group --desired-capacity 3
```

```
fi
```

---

## 14. Environment Setup

This script sets environment variables for different environments (development, staging, production).

```
#!/bin/bash

# Set environment variables for different stages

if [ "$1" == "production" ]; then

    export DB_HOST="prod-db.example.com"

    export API_KEY="prod-api-key"

elif [ "$1" == "staging" ]; then

    export DB_HOST="staging-db.example.com"

    export API_KEY="staging-api-key"

else

    export DB_HOST="dev-db.example.com"

    export API_KEY="dev-api-key"

fi
```

---

## 15. Error Handling and Alerts

This script checks logs for errors and sends a Slack notification if an error is found.

```
#!/bin/bash

# Check logs for error messages and send Slack notification

if grep -i "error" /var/log/myapp.log; then
```

```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Error found in logs!"}' https://hooks.slack.com/services/your/webhook/url
```

```
fi
```

---

## 16. Automated Software Installation and Updates

This script installs Docker if it's not already installed on the system.

```
#!/bin/bash
```

### **# Install Docker**

```
if ! command -v docker &> /dev/null; then
```

```
    curl -fsSL https://get.docker.com -o get-docker.sh
```

```
    sudo sh get-docker.sh
```

```
fi
```

---

## 17. Configuration Management

This script updates configuration files (like nginx.conf) across multiple servers.

```
#!/bin/bash
```

### **# Update nginx configuration across all servers**

```
scp nginx.conf user@server:/etc/nginx/nginx.conf
```

```
ssh user@server "systemctl restart nginx"
```

---

## 18. Health Check Automation

This script checks the health of multiple web servers by making HTTP requests.

```
#!/bin/bash
```

```
# Check if web servers are running
```

```
for server in "server1" "server2" "server3"; do
```

```
    curl -s --head http://$server | head -n 1 | grep "HTTP/1.1 200 OK" > /dev/null
```

```
    if [ $? -ne 0 ]; then
```

```
        echo "$server is down"
```

```
    else
```

```
        echo "$server is up"
```

```
    fi
```

```
done
```

---

## 19. Automated Cleanup of Temporary Files

This script removes files older than 30 days from the /tmp directory to free up disk space.

```
#!/bin/bash
```

```
# Remove files older than 30 days in /tmp
```

```
find /tmp -type f -mtime +30 -exec rm -f {} \;
```

---

## 20. Environment Variable Management

This script sets environment variables from a .env file.

```
#!/bin/bash
```

```
# Set environment variables from a .env file
```

```
export $(grep -v '^#' .env | xargs)
```

---

## 21. Server Reboot Automation

This script automatically reboots the server during off-hours (between 2 AM and 4 AM).

```
#!/bin/bash
```



### **# Reboot server during off-hours**

```
if [ $(date +%H) -ge 2 ] && [ $(date +%H) -lt 4 ]; then  
    sudo reboot  
fi
```

---

## **22. SSL Certificate Renewal**

This script renews SSL certificates using certbot and reloads the web server.

```
#!/bin/bash  
  
# Renew SSL certificates using certbot  
  
certbot renew  
  
systemctl reload nginx
```

---

## **23. Automatic Scaling of Containers**

This script checks the CPU usage of a Docker container and scales it based on usage.

```
#!/bin/bash  
  
# Check CPU usage of a Docker container and scale if necessary  
  
CPU_USAGE=$(docker stats --no-stream --format "{{.CPUPerc}}" my-container |  
sed 's/%//')
```

```
if (( $(echo "$CPU_USAGE > 80" | bc -l) )); then  
    docker-compose scale my-container=3  
fi
```

---

## 24. Backup Verification

This script verifies the integrity of backup files and reports any corrupted ones.

```
#!/bin/bash  
  
# Verify backup files integrity  
for backup in /backups/*.tar.gz; do  
    if ! tar -tzf $backup > /dev/null 2>&1; then  
        echo "Backup $backup is corrupted"  
    else  
        echo "Backup $backup is valid"  
    fi  
done
```

---

## 25. Automated Server Cleanup

This script removes unused Docker images, containers, and volumes to save disk space.

```
#!/bin/bash
```

```
# Remove unused Docker images, containers, and volumes
```

```
docker system prune -af
```

---

## 26. Version Control Operations

This script pulls the latest changes from a Git repository and creates a release tag.

```
#!/bin/bash
```

```
# Pull latest changes from Git repository and create a release tag
```

```
git pull origin main
```

```
git tag -a v$(date +%Y%m%d%H%M%S) -m "Release $(date)"
```

```
git push origin --tags
```

---

## 27. Application Deployment Rollback

This script reverts to the previous Docker container image if a deployment fails.

```
#!/bin/bash
```

```
# Rollback to the previous Docker container image if deployment fails
```

```
if [ $? -ne 0 ]; then  
    docker-compose down  
    docker-compose pull my-app:previous  
    docker-compose up -d  
fi
```

---

## 28. Automated Log Collection

This script collects logs from multiple servers and uploads them to an S3 bucket.

```
#!/bin/bash
```

```
# Collect logs and upload them to an S3 bucket
```

```
tar -czf /tmp/logs.tar.gz /var/log/*
```

```
aws s3 cp /tmp/logs.tar.gz s3://my-log-bucket/logs/$(date  
+%Y%m%d%H%M%S).tar.gz
```

---

## 29. Security Patch Management

This script checks for available security patches and applies them automatically.

```
#!/bin/bash
```

```
# Check and apply security patches  
  
sudo apt-get update  
  
sudo apt-get upgrade -y --only-upgrade
```

---

## 30. Custom Monitoring Scripts

This script checks if a database service is running and restarts it if necessary.

```
#!/bin/bash  
  
# Check if a database service is running and restart it if necessary  
  
if ! systemctl is-active --quiet mysql; then  
  
    systemctl restart mysql  
  
    echo "MySQL service was down and has been restarted"  
  
else  
  
    echo "MySQL service is running"  
  
fi
```

---

## 31. DNS Configuration Automation (Route 53)

```
#!/bin/bash
```

## **# Variables**

ZONE\_ID="your-hosted-zone-id"

DOMAIN\_NAME="your-domain.com"

NEW\_IP="your-new-ip-address"

## **# Update Route 53 DNS record**

```
aws route53 change-resource-record-sets --hosted-zone-id $ZONE_ID
--change-batch '{
```

```
"Changes": [
```

```
{
```

```
  "Action": "UPSERT",
```

```
  "ResourceRecordSet": {
```

```
    "Name": "$DOMAIN_NAME",
```

```
    "Type": "A",
```

```
    "TTL": 60,
```

```
    "ResourceRecords": [
```

```
      {
```

```
        "Value": "$NEW_IP"
```

```
      }
```

```
    ]
```

```
  }
```

```
}  
]  
'
```

---

## 32. Automated Code Linting and Formatting (ESLint and Prettier)

```
#!/bin/bash
```

```
# Run ESLint
```

```
npx eslint . --fix
```

```
# Run Prettier
```

```
npx prettier --write "**/*.js"
```

---

## 33. Automated API Testing (Using curl)

```
#!/bin/bash
```

```
# API URL
```

```
API_URL="https://your-api-endpoint.com/endpoint"
```

```
# Make GET request and check for 200 OK response
```

```
RESPONSE=$(curl --write-out "%{http_code}" --silent --output /dev/null  
$API_URL)
```

```
if [ $RESPONSE -eq 200 ]; then
```

```
    echo "API is up and running"
```

```
else
```

```
    echo "API is down. Response code: $RESPONSE"
```

```
fi
```

---

## 34. Container Image Scanning (Using Trivy)

```
#!/bin/bash
```

```
# Image to scan
```

```
IMAGE_NAME="your-docker-image:latest"
```

```
# Run Trivy scan
```

```
trivy image --exit-code 1 --severity HIGH,CRITICAL $IMAGE_NAME
```



```
if [ $? -eq 1 ]; then
    echo "Vulnerabilities found in image: $IMAGE_NAME"
    exit 1
else
    echo "No vulnerabilities found in image: $IMAGE_NAME"
fi
```

---

## 35. Disk Usage Monitoring and Alerts (Email Notification)

```
#!/bin/bash
```

```
# Disk usage threshold
```

```
THRESHOLD=80
```

```
# Get current disk usage percentage
```

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

```
# Check if disk usage exceeds threshold
```

```
if [ $DISK_USAGE -gt $THRESHOLD ]; then
```

```
    echo "Disk usage is above threshold: $DISK_USAGE%" | mail -s "Disk Usage Alert" your-email@example.com
```

fi

---

## 36. Automated Load Testing (Using Apache Benchmark)

```
#!/bin/bash
```

```
# Target URL
```

```
URL="https://your-application-url.com"
```

```
# Run Apache Benchmark with 1000 requests and 10 concurrent requests
```

```
ab -n 1000 -c 10 $URL
```

---

## 37. Automated Email Reports (Server Health Report)

```
#!/bin/bash
```

```
# Server Health Report
```

```
REPORT=$(top -n 1 | head -n 10)
```

```
# Send report via email
```

```
echo "$REPORT" | mail -s "Server Health Report" your-email@example.com
```

---

## 38. DNS Configuration Automation (Route 53)

**Introduction:** This script automates the process of updating DNS records in AWS Route 53 when the IP address of a server changes. It ensures that DNS records are updated dynamically when new servers are provisioned.

```
#!/bin/bash
```

### # Variables

```
ZONE_ID="your-hosted-zone-id"
```

```
DOMAIN_NAME="your-domain.com"
```

```
NEW_IP="your-new-ip-address"
```

### # Update Route 53 DNS record

```
aws route53 change-resource-record-sets --hosted-zone-id $ZONE_ID  
--change-batch '{
```

```
  "Changes": [
```

```
    {
```

```
      "Action": "UPSERT",
```

```
      "ResourceRecordSet": {
```

```
        "Name": "$DOMAIN_NAME",
```

```
"Type": "A",  
"TTL": 60,  
"ResourceRecords": [  
  {  
    "Value": "$NEW_IP"  
  }  
]  
}  
}  
]  
'
```

---

## 39. Automated Code Linting and Formatting (ESLint and Prettier)

**Introduction:** This script runs ESLint and Prettier to check and automatically format JavaScript code before deployment. It ensures code quality and consistency.

```
#!/bin/bash
```

```
# Run ESLint
```

```
npx eslint . --fix
```

## **# Run Prettier**

```
npx prettier --write "**/*.js"
```

---

## **40. Automated API Testing (Using curl)**

**Introduction:** This script automates the process of testing an API by sending HTTP requests and verifying the response status. It helps ensure that the API is functioning correctly.

```
#!/bin/bash
```

### **# API URL**

```
API_URL="https://your-api-endpoint.com/endpoint"
```

```
# Make GET request and check for 200 OK response
```

```
RESPONSE=$(curl --write-out "%{http_code}" --silent --output /dev/null  
$API_URL)
```

```
if [ $RESPONSE -eq 200 ]; then
```

```
    echo "API is up and running"
```

```
else
```

```
    echo "API is down. Response code: $RESPONSE"
```

fi

---

## 41. Container Image Scanning (Using Trivy)

**Introduction:** This script scans Docker images for known vulnerabilities using Trivy. It ensures that only secure images are deployed in production.

```
#!/bin/bash
```

```
# Image to scan
```

```
IMAGE_NAME="your-docker-image:latest"
```

```
# Run Trivy scan
```

```
trivy image --exit-code 1 --severity HIGH,CRITICAL $IMAGE_NAME
```

```
if [ $? -eq 1 ]; then
```

```
    echo "Vulnerabilities found in image: $IMAGE_NAME"
```

```
    exit 1
```

```
else
```

```
    echo "No vulnerabilities found in image: $IMAGE_NAME"
```

```
fi
```

---

## 42. Disk Usage Monitoring and Alerts (Email Notification)

**Introduction:** This script monitors disk usage and sends an alert via email if the disk usage exceeds a specified threshold. It helps in proactive monitoring of disk space.

```
#!/bin/bash
```

```
# Disk usage threshold
```

```
THRESHOLD=80
```

```
# Get current disk usage percentage
```

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

```
# Check if disk usage exceeds threshold
```

```
if [ $DISK_USAGE -gt $THRESHOLD ]; then
```

```
    echo "Disk usage is above threshold: $DISK_USAGE%" | mail -s "Disk Usage Alert" your-email@example.com
```

```
fi
```

---

## 43. Automated Load Testing (Using Apache Benchmark)

**Introduction:** This script runs load tests using Apache Benchmark (ab) to simulate traffic on an application. It helps measure the performance and scalability of the application.

```
bash
```

```
#!/bin/bash
```

```
# Target URL
```

```
URL="https://your-application-url.com"
```

```
# Run Apache Benchmark with 1000 requests and 10 concurrent requests
```

```
ab -n 1000 -c 10 $URL
```

---

## 44. Automated Email Reports (Server Health Report)

**Introduction:** This script generates a server health report using system commands like top and sends it via email. It helps keep track of server performance and health.

```
#!/bin/bash
```

```
# Server Health Report
```

```
REPORT=$(top -n 1 | head -n 10)
```

```
# Send report via email
```

```
echo "$REPORT" | mail -s "Server Health Report" your-email@example.com
```



---

## 45. Automating Documentation Generation (Using pdoc for Python)

**Introduction:** This script generates HTML documentation from Python code using pdoc. It helps automate the process of creating up-to-date documentation from the source code.

```
#!/bin/bash
```

```
# Generate documentation using pdoc
```

```
pdoc --html your-python-module --output-dir docs/
```

```
# Optionally, you can zip the generated docs
```

```
zip -r docs.zip docs/
```

---

# Automation with Python for DevOps

## 1. File Operations

**Read a file:**

```
python
```

```
with open('file.txt', 'r') as file:
```

```
    content = file.read()
```

```
    print(content)
```

### **Write to a file:**

```
python
```

```
with open('output.txt', 'w') as file:
```

```
    file.write('Hello, DevOps!')
```

---

## **2. Environment Variables**

### **Get an environment variable:**

```
python
```

```
import os
```

```
db_user = os.getenv('DB_USER')
```

```
print(db_user)
```

**Set an environment variable:**

```
python
```

```
import os
```

```
os.environ['NEW_VAR'] = 'value'
```

---

### **3. Subprocess Management**

**Run shell commands:**

```
python
```

```
import subprocess
```

```
result = subprocess.run(['ls', '-l'], capture_output=True, text=True)
```

```
print(result.stdout)
```

---

### **4. API Requests**

**Make a GET request:**

```
python
```

```
import requests
```

```
response = requests.get('https://api.example.com/data')
```

```
print(response.json())
```

---

## 5. JSON Handling

### Read JSON from a file:

```
python
```

```
import json
```

```
with open('data.json', 'r') as file:
```

```
    data = json.load(file)
```

```
    print(data)
```

### Write JSON to a file:

```
python
```

```
import json
```

```
data = {'name': 'DevOps', 'type': 'Workflow'}
```

```
with open('output.json', 'w') as file:
```

```
    json.dump(data, file, indent=4)
```

---

## 6. Logging

Logging improves visibility, helps resolve issues faster, and optimizes system performance.

### Basic logging setup:

```
python
```

```
import logging
```

```
logging.basicConfig(level=logging.INFO)
```

```
logging.info('This is an informational message')
```

---

## 7. Working with Databases

### Connect to a SQLite database:

```
python
```

```
import sqlite3
```

```
conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute('CREATE TABLE IF NOT EXISTS users (id INTEGER
PRIMARY KEY, name TEXT)')

conn.commit()

conn.close()
```

---

## 8. Automation with Libraries

### Using Paramiko for SSH connections:

```
python

import paramiko

ssh = paramiko.SSHClient()

ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

ssh.connect('hostname', username='user', password='password')

stdin, stdout, stderr = ssh.exec_command('ls')

print(stdout.read().decode())

ssh.close()
```

---

## 9. Error Handling

### Try-except block:

python

try:

    # code that may raise an exception

    risky\_code()

except Exception as e:

    print(f'Error occurred: {e}')

---

## 10. Docker Integration

### Using the docker package to interact with Docker:

python

import docker

client = docker.from\_env()

containers = client.containers.list()

for container in containers:

```
    print(container.name)
```

---

## 11. Working with YAML Files

### Read a YAML file:

python

```
import yaml
```

with open('config.yaml', 'r') as file:

```
    config = yaml.safe_load(file)
```

```
    print(config)
```

### Write to a YAML file:

python

```
import yaml
```

```
data = {'name': 'DevOps', 'version': '1.0'}
```

with open('output.yaml', 'w') as file:

```
    yaml.dump(data, file)
```



---

## 12. Parsing Command-Line Arguments

**Using argparse:**

```
python
```

```
import argparse
```

```
parser = argparse.ArgumentParser(description='Process some integers.')
```

```
parser.add_argument('--num', type=int, help='an integer for the accumulator')
```

```
args = parser.parse_args()
```

```
print(args.num)
```

---

## 13. Monitoring System Resources

**Using psutil to monitor system resources:**

```
python
```

```
import psutil
```

```
print(f'CPU Usage: {psutil.cpu_percent()}%')
```

```
print(f'Memory Usage: {psutil.virtual_memory().percent}%')
```

---

## 14. Handling HTTP Requests with Flask

### Basic Flask API:

```
python

from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/health', methods=['GET'])
def health_check():
    return jsonify({'status': 'healthy'})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

---

## 15. Creating Docker Containers

### Using the Docker SDK to create a container:

```
python

import docker
```

```
client = docker.from_env()
container = client.containers.run('ubuntu', 'echo Hello World', detach=True)
print(container.logs())
```

---

## 16. Scheduling Tasks

**Using schedule for task scheduling:**

```
python
```

```
import schedule
```

```
import time
```

```
def job():
```

```
    print("Running scheduled job...")
```

```
schedule.every(1).minutes.do(job)
```

```
while True:
```

```
    schedule.run_pending()
```

```
    time.sleep(1)
```

## 17. Version Control with Git

**Using GitPython to interact with Git repositories:**

```
python

import git

repo = git.Repo('/path/to/repo')

repo.git.add('file.txt')

repo.index.commit('Added file.txt')
```

---

## 18. Email Notifications

**Sending emails using smtplib:**

```
python

import smtplib

from email.mime.text import MIMEText

msg = MIMEText('This is the body of the email')

msg['Subject'] = 'Email Subject'

msg['From'] = 'you@example.com'

msg['To'] = 'recipient@example.com'
```

with smtplib.SMTP('smtp.example.com', 587) as server:

```
server.starttls()
```

```
server.login('your_username', 'your_password')
```

```
server.send_message(msg)
```

---

## 19. Creating Virtual Environments

**Creating and activating a virtual environment:**

```
python
```

```
import os
```

```
import subprocess
```

```
# Create virtual environment
```

```
subprocess.run(['python3', '-m', 'venv', 'myenv'])
```

```
# Activate virtual environment (Windows)
```

```
os.system('myenv\\Scripts\\activate')
```

```
# Activate virtual environment (Linux/Mac)
```

```
os.system('source myenv/bin/activate')
```

---

## 20. Integrating with CI/CD Tools

Using the requests library to trigger a Jenkins job:

```
python
```

```
import requests
```

```
url = 'http://your-jenkins-url/job/your-job-name/build'
```

```
response = requests.post(url, auth=('user', 'token'))
```

```
print(response.status_code)
```

---

## 21. Using watchdog for File System Monitoring

Monitor changes in a directory.

```
python
```

```
from watchdog.observers import Observer
```

```
from watchdog.events import FileSystemEventHandler
```

```
import time
```

```
class MyHandler(FileSystemEventHandler):

    def on_modified(self, event):

        print(f'File modified: {event.src_path}')

event_handler = MyHandler()

observer = Observer()

observer.schedule(event_handler, path='path/to/monitor', recursive=False)

observer.start()

try:

    while True:

        time.sleep(1)

except KeyboardInterrupt:

    observer.stop()

observer.join()
```

---

## 22. Testing Code

**Using unittest for unit testing:**

python

```
import unittest
```

```
def add(a, b):  
    return a + b  
  
class TestMathFunctions(unittest.TestCase):  
    def test_add(self):  
        self.assertEqual(add(2, 3), 5)  
  
if __name__ == '__main__':  
    unittest.main()
```

---

## 23. Data Transformation with Pandas

**Using pandas for data manipulation:**

python

```
import pandas as pd  
  
df = pd.read_csv('data.csv')  
df['new_column'] = df['existing_column'] * 2  
df.to_csv('output.csv', index=False)
```



---

## 24. Using Python for Infrastructure as Code

### Using boto3 for AWS operations:

```
python
```

```
import boto3
```

```
ec2 = boto3.resource('ec2')
```

```
instances = ec2.instances.filter(Filters=[{'Name': 'instance-state-name', 'Values':  
['running']}])
```

```
for instance in instances:
```

```
    print(instance.id, instance.state)
```

---

## 25. Web Scrapping

Web scraping with BeautifulSoup allows you to extract data from websites programmatically. You can use it to automatically gather information like text, links, images, or titles from web pages. This is useful for tasks like monitoring prices, gathering research data, or tracking changes on websites. It saves time compared to manually collecting data and can be customized to fit specific needs.

### Using BeautifulSoup to scrape web pages:

```
python
```

```
import requests
```

```
from bs4 import BeautifulSoup

response = requests.get('http://example.com')

soup = BeautifulSoup(response.content, 'html.parser')

print(soup.title.string)
```

## **26. Using Fabric for Remote Execution**

### **Running commands on a remote server:**

```
python
```

```
from fabric import Connection
```

```
conn = Connection(host='user@hostname', connect_kwargs={'password':
'your_password'})

conn.run('uname -s')
```

---

## **27. Automating AWS S3 Operations**

### **Upload and download files using boto3:**

```
python
```

```
import boto3
```

```
s3 = boto3.client('s3')
```

#### **# Upload a file**

```
s3.upload_file('local_file.txt', 'bucket_name', 's3_file.txt')
```

#### **# Download a file**

```
s3.download_file('bucket_name', 's3_file.txt', 'local_file.txt')
```

---

## **28. Monitoring Application Logs**

**Tail logs using tail -f equivalent in Python:**

```
python
```

```
import time
```

```
def tail_f(file):
```

```
    file.seek(0, 2) # Move to the end of the file
```

```
    while True:
```

```
        line = file.readline()
```

```
        if not line:
```

```
        time.sleep(0.1) # Sleep briefly

        continue

    print(line)

with open('app.log', 'r') as log_file:

    tail_f(log_file)
```

---

## 29. Container Health Checks

**Check the health of a running Docker container:**

```
python

import docker

client = docker.from_env()

container = client.containers.get('container_id')

print(container.attrs['State']['Health']['Status'])
```

---

## 30. Using requests for Rate-Limited APIs

**Handle rate limiting in API requests:**

```
python

import requests
```

```
import time

url = 'https://api.example.com/data'

while True:

    response = requests.get(url)

    if response.status_code == 200:

        print(response.json())

        break

    elif response.status_code == 429: # Too Many Requests

        time.sleep(60) # Wait a minute before retrying

    else:

        print('Error:', response.status_code)

        break
```

---

## 31. Docker Compose Integration

**Using docker-compose in Python:**

```
python

import os

import subprocess
```

**# Start services defined in docker-compose.yml**

```
subprocess.run(['docker-compose', 'up', '-d'])
```

**# Stop services**

```
subprocess.run(['docker-compose', 'down'])
```

---

## **46. Creating a REST API with Flask-RESTful**

A simple REST API that returns a "hello world" message.

python

```
from flask import Flask
```

```
from flask_restful import Resource, Api
```

```
app = Flask(__name__)
```

```
api = Api(app)
```

```
class HelloWorld(Resource):
```

```
    def get(self):
```

```
        return {'hello': 'world'}
```

```
api.add_resource>HelloWorld, '/')
```

```
if __name__ == '__main__':
```

```
    app.run(debug=True)
```

---

## 47. Using asyncio for Asynchronous Tasks

Using **asyncio** in DevOps improves efficiency by allowing concurrent execution of I/O-bound tasks (like interacting with APIs or services) without blocking. It helps scale operations, speeds up workflows (such as deployments or health checks), and optimizes resource usage. This is especially useful in CI/CD pipelines for faster, non-blocking automation.

Running asynchronous tasks in Python.

```
python
```

```
import asyncio
```

```
async def main():
```

```
    print('Hello')
```

```
    await asyncio.sleep(1)
```

```
    print('World')
```

```
asyncio.run(main())
```

---

## 48. Network Monitoring with scapy

Packet sniffing using scapy.

python

```
from scapy.all import sniff
```

```
def packet_callback(packet):
```

```
    print(packet.summary())
```

```
sniff(prn=packet_callback, count=10)
```

---

## 49. Handling Configuration Files with configparser

Reading and writing to INI configuration files.

python

```
import configparser
```

```
config = configparser.ConfigParser()
```

```
config.read('config.ini')
```



```
print(config['DEFAULT']['SomeSetting'])
```

```
config['DEFAULT']['NewSetting'] = 'Value'
```

```
with open('config.ini', 'w') as configfile:
```

```
    config.write(configfile)
```

---

## 50. WebSocket Client Example

Creating a WebSocket client with websocket-client.

python

```
import websocket
```

```
def on_message(ws, message):
```

```
    print("Received message:", message)
```

```
ws = websocket.WebSocketApp("ws://echo.websocket.org",  
on_message=on_message)
```

```
ws.run_forever()
```

---

## 51. Creating a Docker Image with Python

Using docker library to build an image.

```
python
```

```
import docker
```

```
client = docker.from_env()
```

```
# Dockerfile content
```

```
dockerfile_content = """
```

```
FROM python:3.9-slim
```

```
WORKDIR /app
```

```
COPY . /app
```

```
RUN pip install -r requirements.txt
```

```
CMD ["python", "app.py"]
```

```
"""
```

```
# Create a Docker image
```

```
image, build_logs = client.images.build(fileobj=dockerfile_content.encode('utf-8'),  
tag='my-python-app')
```

```
for line in build_logs:
```

```
    print(line)
```

---

## 52. Using psutil for System Monitoring

Retrieve system metrics such as CPU and memory usage.

```
python
```

```
import psutil
```

```
print("CPU Usage:", psutil.cpu_percent(interval=1), "%")
```

```
print("Memory Usage:", psutil.virtual_memory().percent, "%")
```

---

## 53. Database Migration with Alembic

Script to initialize Alembic migrations.

```
python
```

```
from alembic import command
```

```
from alembic import config
```

```
alembic_cfg = config.Config("alembic.ini")
```

```
command.upgrade(alembic_cfg, "head")
```

---

## 54. Using paramiko for SSH Connections

Execute commands on a remote server via SSH.

Paramiko helps you connect to remote servers securely, run commands, and automate tasks using Python. It simplifies managing remote systems by ensuring encrypted connections.

```
python
```

```
import paramiko
```

```
client = paramiko.SSHClient()
```

```
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

```
client.connect('hostname', username='user', password='your_password')
```

```
stdin, stdout, stderr = client.exec_command('ls -la')
```

```
print(stdout.read().decode())
```

```
client.close()
```

---

## 55. CloudFormation Stack Creation with boto3

Creating an AWS CloudFormation stack.

```
python
```

```
import boto3
```

```
cloudformation = boto3.client('cloudformation')
```

```
with open('template.yaml', 'r') as template_file:
```

```
    template_body = template_file.read()
```

```
response = cloudformation.create_stack(
```

```
    StackName='MyStack',
```

```
    TemplateBody=template_body,
```

```
    Parameters=[{'ParameterKey': 'InstanceType', 'ParameterValue': 't2.micro'}],
```

```
    TimeoutInMinutes=5,
```

```
    Capabilities=['CAPABILITY_NAMED_IAM'],
```

```
)
```

```
print(response)
```

## **56. Automating EC2 Instance Management**

Starting and stopping EC2 instances.

```
python
```

```
import boto3
```

```
ec2 = boto3.resource('ec2')
```

```
# Start an instance
```

```
instance = ec2.Instance('instance_id')
```

```
instance.start()
```

```
# Stop an instance
```

```
instance.stop()
```

---

## **57. Automated Backup with shutil**

Backup files to a specific directory.

```
python
```

```
import shutil
```

```
import os
```

```
source_dir = '/path/to/source'
```

```
backup_dir = '/path/to/backup'
```

```
shutil.copytree(source_dir, backup_dir)
```