ADVANCED PYTHON PROJECTS WITH SOLUTIONS

1. Automated AWS Infrastructure with Boto3

Steps:

1. **Install Boto3**: `pip install boto3`
2. **Authenticate with AWS credentials**
3. **Create an EC2 instance using Python**

Python Script (`aws_ec2.py`)

```
python
CopyEdit
import boto3

aws_region = "us-east-1"
instance_type = "t2.micro"
ami_id = "ami-0abcdef1234567890"  # Replace with a valid AMI ID

ec2 = boto3.client("ec2", region_name=aws_region)

response = ec2.run_instances(
    ImageId=ami_id,
    InstanceType=instance_type,
    MinCount=1,
    MaxCount=1,
    KeyName="your-keypair",  # Replace with your key pair
)

print("EC2 Instance Created:", response["Instances"][0]["InstanceId"])
```

Solution:

✅ Automates EC2 instance creation
✅ Reduces manual AWS console work

2. AI-Powered Log Analyzer (NLP)

Steps:

1. **Use Python to read logs**
2. **Analyze errors using NLP (spaCy)**
3. **Summarize frequent issues**

Python Script (`log_analyzer.py`)

```
python
CopyEdit
```

```
import spacy
from collections import Counter

nlp = spacy.load("en_core_web_sm")

with open("/var/log/syslog", "r") as file:
    logs = file.readlines()

log_text = " ".join(logs)
doc = nlp(log_text)

error_counts = Counter(token.text for token in doc if token.text.lower() in
["error", "failed", "critical"])

print("Frequent Log Errors:", error_counts)
```

Solution:

✅ Uses NLP to analyze logs
✅ Helps DevOps teams detect critical issues

3. Python CI/CD Pipeline with GitHub Actions

Steps:

1. **Create a GitHub Actions workflow (`.github/workflows/main.yml`)**
2. **Automate build, test, and deployment**

GitHub Actions Workflow (`main.yml`)

```
yaml
CopyEdit
name: Python CI/CD

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v3
        with:
          python-version: "3.9"

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Run tests
        run: pytest
```

Solution:

✅ Automates CI/CD pipeline
✅ Runs tests before deployment

4. Network Scanner with Python

Steps:

1. **Use Scapy to scan networks**
2. **Detect active devices**

Python Script (`network_scanner.py`)

```
python
CopyEdit
import scapy.all as scapy

def scan(ip_range):
    arp_request = scapy.ARP(pdst=ip_range)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    packet = broadcast / arp_request
    answered_list = scapy.srp(packet, timeout=1, verbose=False)[0]

    devices = []
    for sent, received in answered_list:
        devices.append({"IP": received.psrc, "MAC": received.hwsrc})

    return devices

network_devices = scan("192.168.1.1/24")
for device in network_devices:
    print(device)
```

Solution:

✅ Scans network for connected devices
✅ Useful for security audits

5. Kubernetes Pod Autoscaler with Python

Steps:

1. **Monitor CPU usage**
2. **Scale pods based on load**

Python Script (`k8s_autoscale.py`)

```
python
CopyEdit
from kubernetes import client, config

config.load_kube_config()
v1 = client.CoreV1Api()
hpa = client.AutoscalingV1Api()

hpa_spec = client.V1HorizontalPodAutoscaler(
    metadata=client.V1ObjectMeta(name="my-app-hpa"),
    spec=client.V1HorizontalPodAutoscalerSpec(
        scale_target_ref=client.V1CrossVersionObjectReference(
            api_version="apps/v1", kind="Deployment", name="my-app"
        ),
        min_replicas=1,
        max_replicas=5,
        target_cpu_utilization_percentage=50
    )
)

hpa.create_namespaced_horizontal_pod_autoscaler(namespace="default",
body=hpa_spec)
print("Autoscaler Created!")
```

Solution:

✅ Automates Kubernetes scaling
✅ Ensures optimal resource usage