

## Key Takeaways

# Automation with Python

# Automate Maintenance Tasks on AWS



## Automated Infrastructure Provisioning

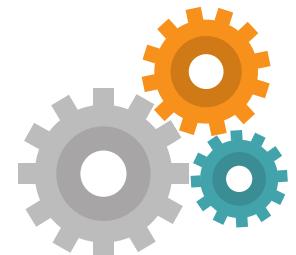
- Automated **creating EC2 instances** with infrastructure resources
- Automated **EKS cluster creation**



- In a big company with hundreds of servers, we would want to automate maintenance work for these servers

With Python we now want to **automate repetitive tasks** like:

- doing regular **back-ups**
- doing regular **clean-ups**
- configuration on existing servers
- doing **health-checks/monitoring**



# Boto3 - AWS Library

Boto3

- A library or package is a reusable chunk of code that you can use in your own programs
- Boto3 specifically is the **AWS SDK to create, configure and manage AWS services**, like EC2, S3, etc.

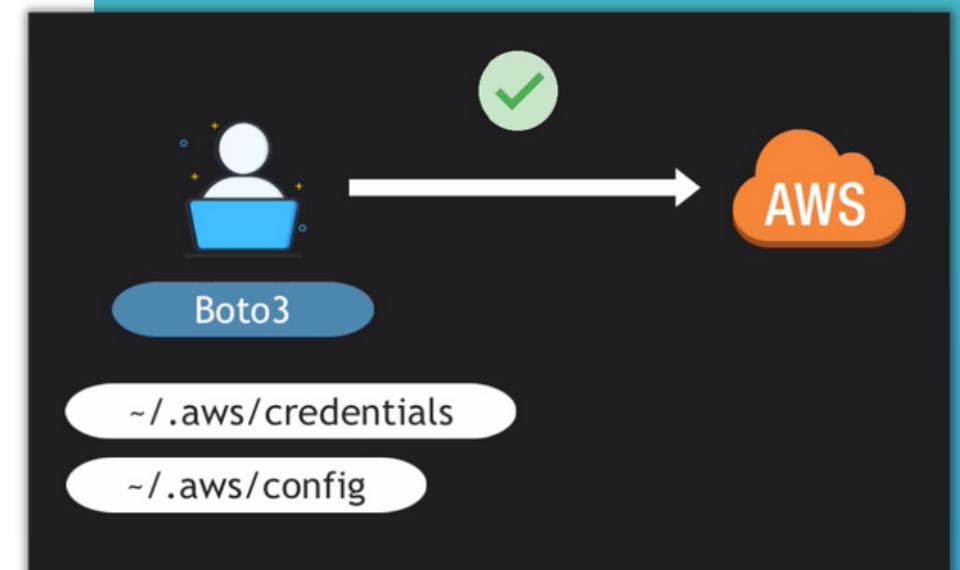


- If you need to communicate with Azure or Google Cloud, there are different libraries specific to that cloud which you can use

## Connect to Boto3

- Before using Boto3, you need to **set up authentication credentials** for your AWS account
- The library provides an object-oriented API as well as low-level access to AWS services:

```
ec2_client = boto3.client('ec2', region_name="eu-central-1")
ec2_resource = boto3.resource('ec2', region_name="eu-central-1")
```



# Terraform vs Python

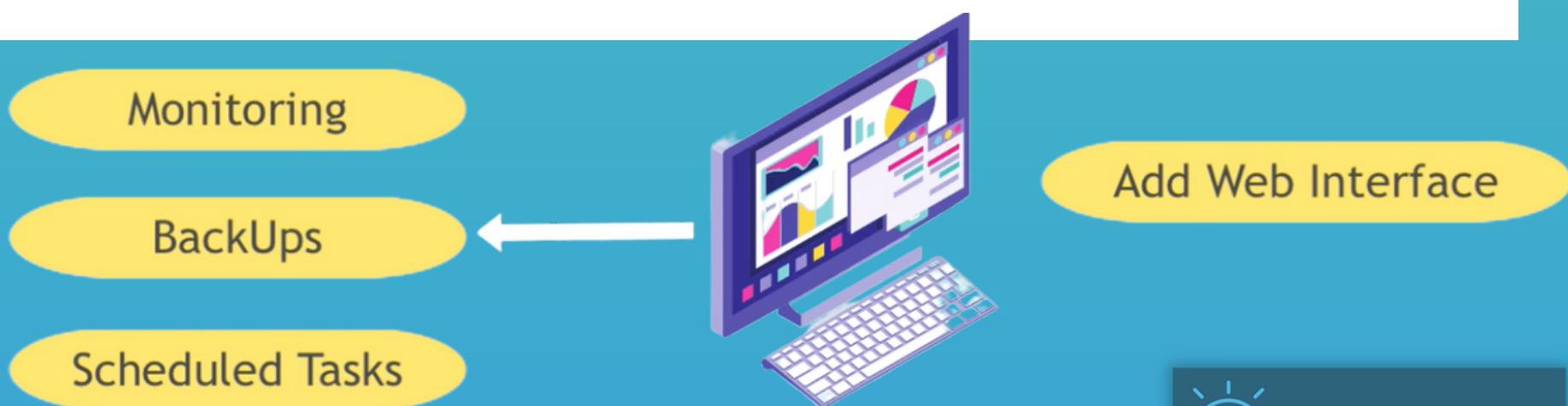
Python can also be used for infrastructure provisioning, but Terraform is much better for this use case

## Terraform

- TF **manages state of the infrastructure**, so TF knows the current state
- TF knows the difference between current and your configured/desired state
- TF is **idempotent** (multiple executions of same config file, will always result in same end result)

## Python

- Python doesn't have a state and is not idempotent
- In TF you declare the end result, while in Python you need to **explicitly write what you want to do** (imperative)
- Python is **more low level, so it's more flexible and you can write very complex logic:**



# Automation Tasks - 1

## Demo Project: EC2 Status Checks

- To know which instances are in which state

The screenshot shows a PyCharm interface with a terminal window displaying EC2 instance status output:

```
for status in statuses['Instances']:
    print(status)
    Instance i-0b2327a695d511e7c is running
    Instance i-0cc03054c5922ac8d is terminated
    Instance i-06dc876ce1050c05 is running
    Instance i-06dc876ce1050c05 status is ok and system status is ok
    Instance i-0b2327a695d511e7c status is ok and system status is ok
```

Below the terminal is a CloudWatch Metrics dashboard showing metrics for three EC2 instances. A cursor points to the third instance's status check row in a table:

Name	Instance ID	Instance state	Instance type	Status check
dev-server-three	i-0092fb0d1ea678c6f	Running	t2.micro	Initializing
dev-server-two	i-0de11486f87f9290a	Running	t2.micro	Initializing
dev-server-one	i-0936e85840d5d227f	Running	t2.micro	Initializing

## Demo Project: Scheduling Status Checks

- To get live updates of the status checks we need a **scheduler**, that triggers the program automatically

The screenshot shows a PyCharm code editor with a Python script named `ec2-status-checks.py`. The code uses the `schedule` library to run a task every 5 seconds:

```
for status in statuses['InstanceStatuses']:
    ins_status = status['InstanceState']['Status']
    sys_status = status['SystemStatus']['Status']
    state = status['InstanceState']['Name']
    print(f"Instance {status['InstanceId']} is {state} with instance status {ins_status} and system status {sys_status}\n")

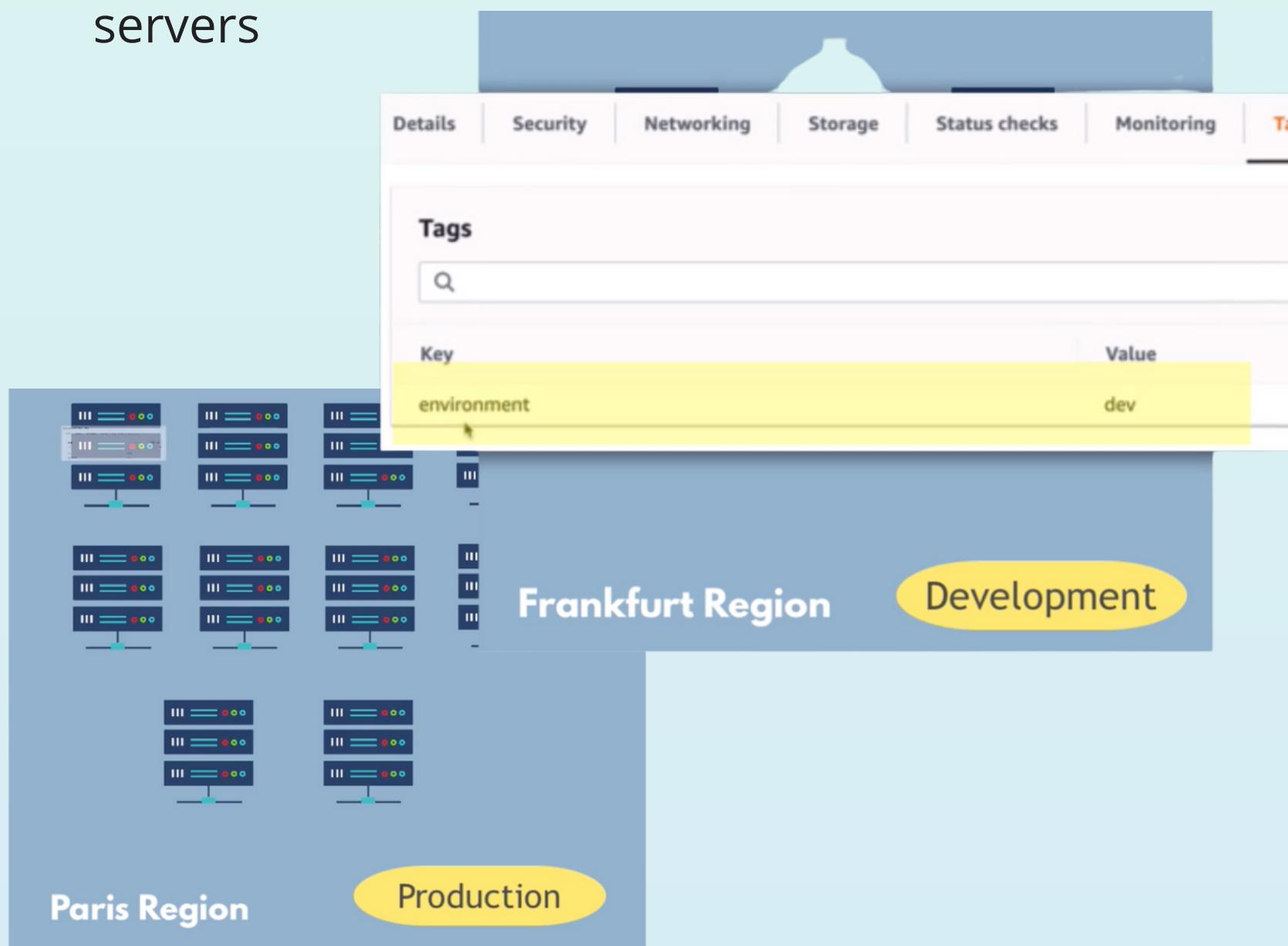
schedule.every(5).seconds.do(check_instance_status)

while True:
    schedule.run_pending()
```

# Automation Tasks - 2

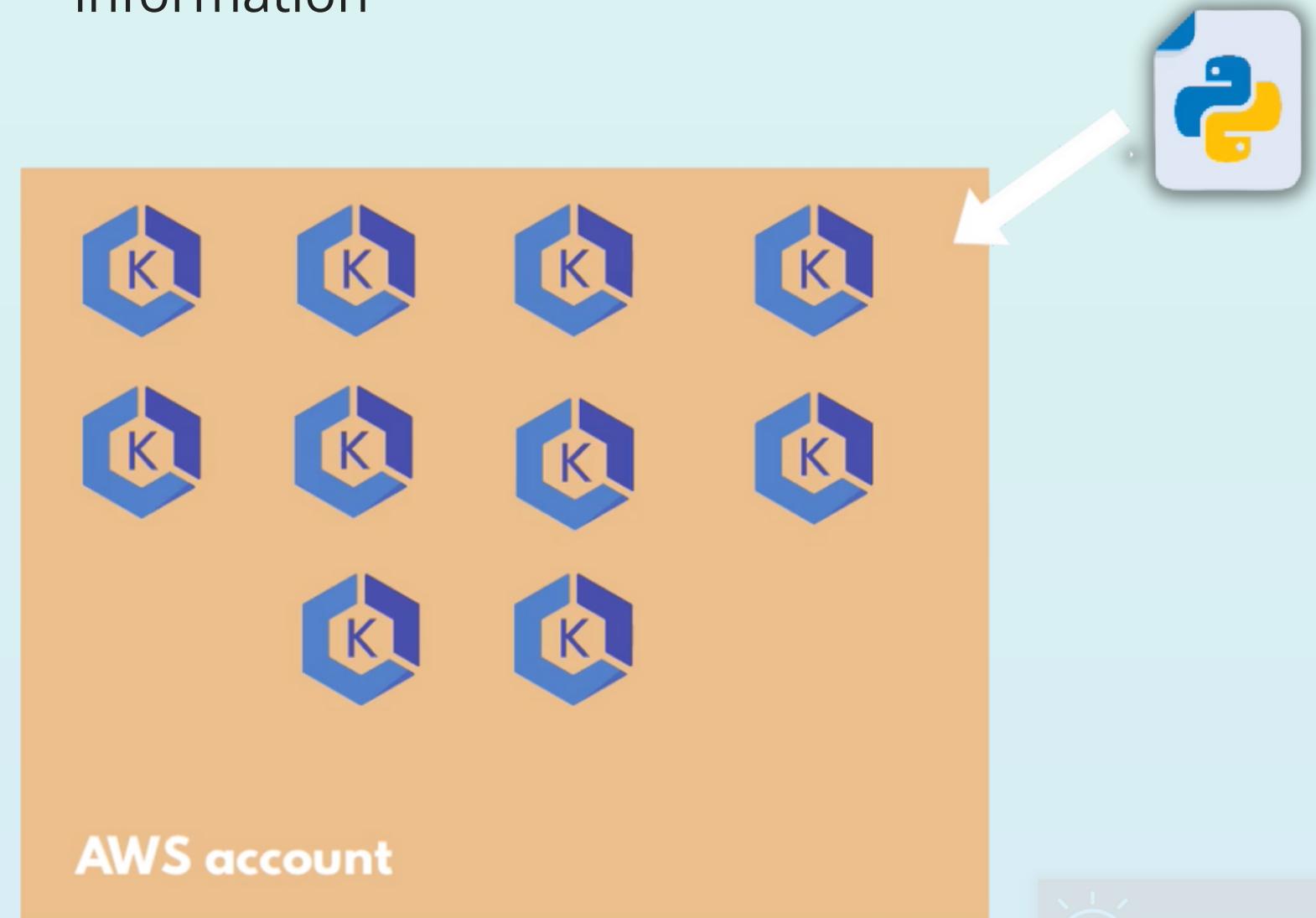
## Demo Project: Add Tags to EC2 Instances

- Having hundreds of instances, you would want to automate adding environment tags for all these servers



## Demo Project: Cluster Information

- If you have 10s or 100s of clusters to manage, you can write a script that gives you all cluster information

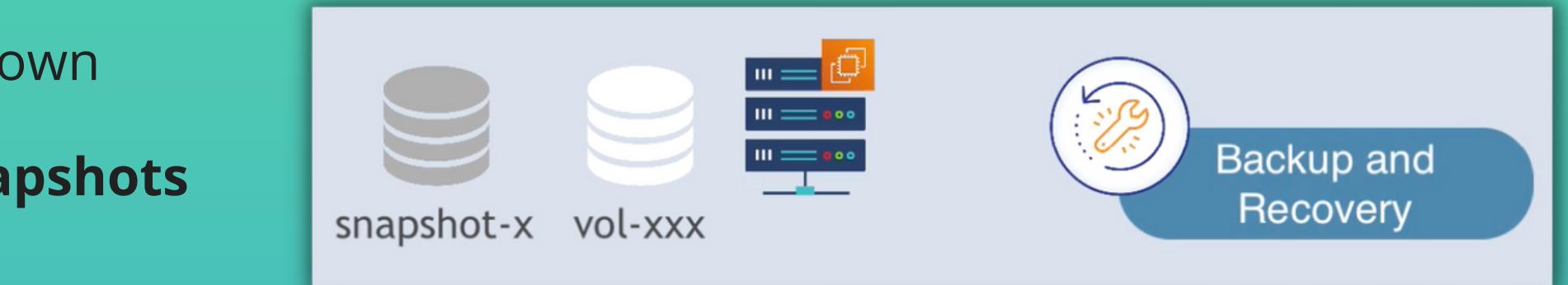


# Automation Tasks - 3

## Demo Project: Automate Data Backup

- Data is stored in AWS Volumes. Each instance has its own volume. To **backup the data, we create volume snapshots (copy of a Volume)**.
- If you have hundreds of instances, you need an automated way to create snapshots:

- We can **schedule regular volume snapshots** with a Python program
- Now if volumes fail or the data in there gets corrupted, we have snapshots to recover



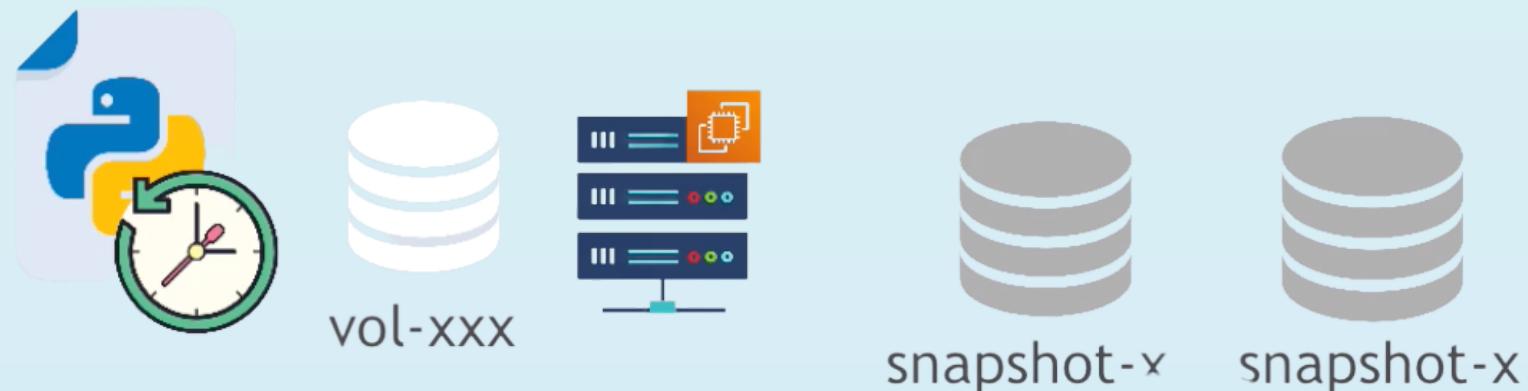
# Automation Tasks - 4

## Demo Project: Automate Snapshot Cleanup

- If Scheduler runs every day, we end up with a lot of snapshots



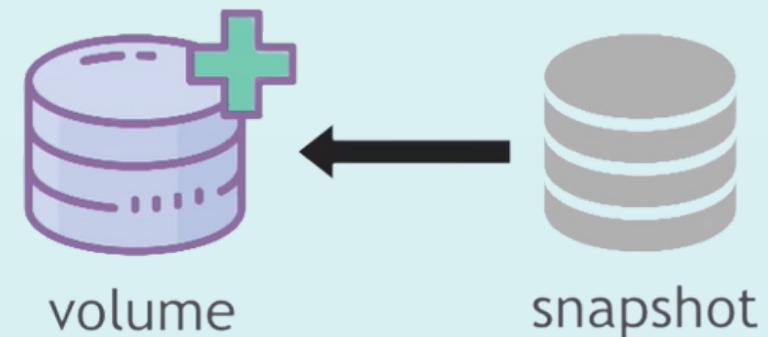
→ We can write a program that **cleans up old snapshots regularly**. E.g. delete all snapshots but the recent 2 snapshots



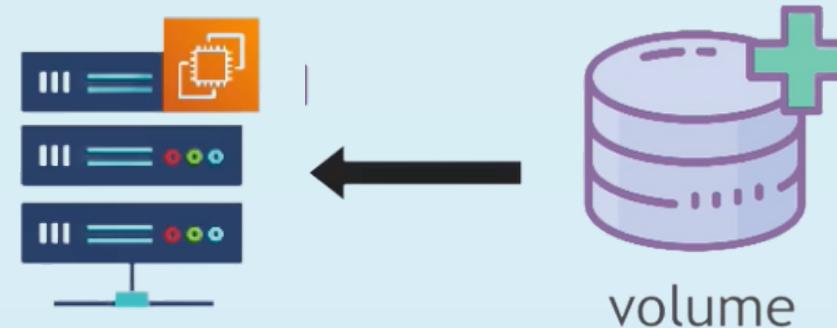
## Demo Project: Automate Restoring EC2 Volumes

- Again if 100s of volumes fail, you **don't want to restore all manually**. So an automated python script is a great use case to recover the latest working state

- 1) Create new Volume from the snapshot



- 2) Attach new Volume to EC2 instance



# Automation Tasks - 5

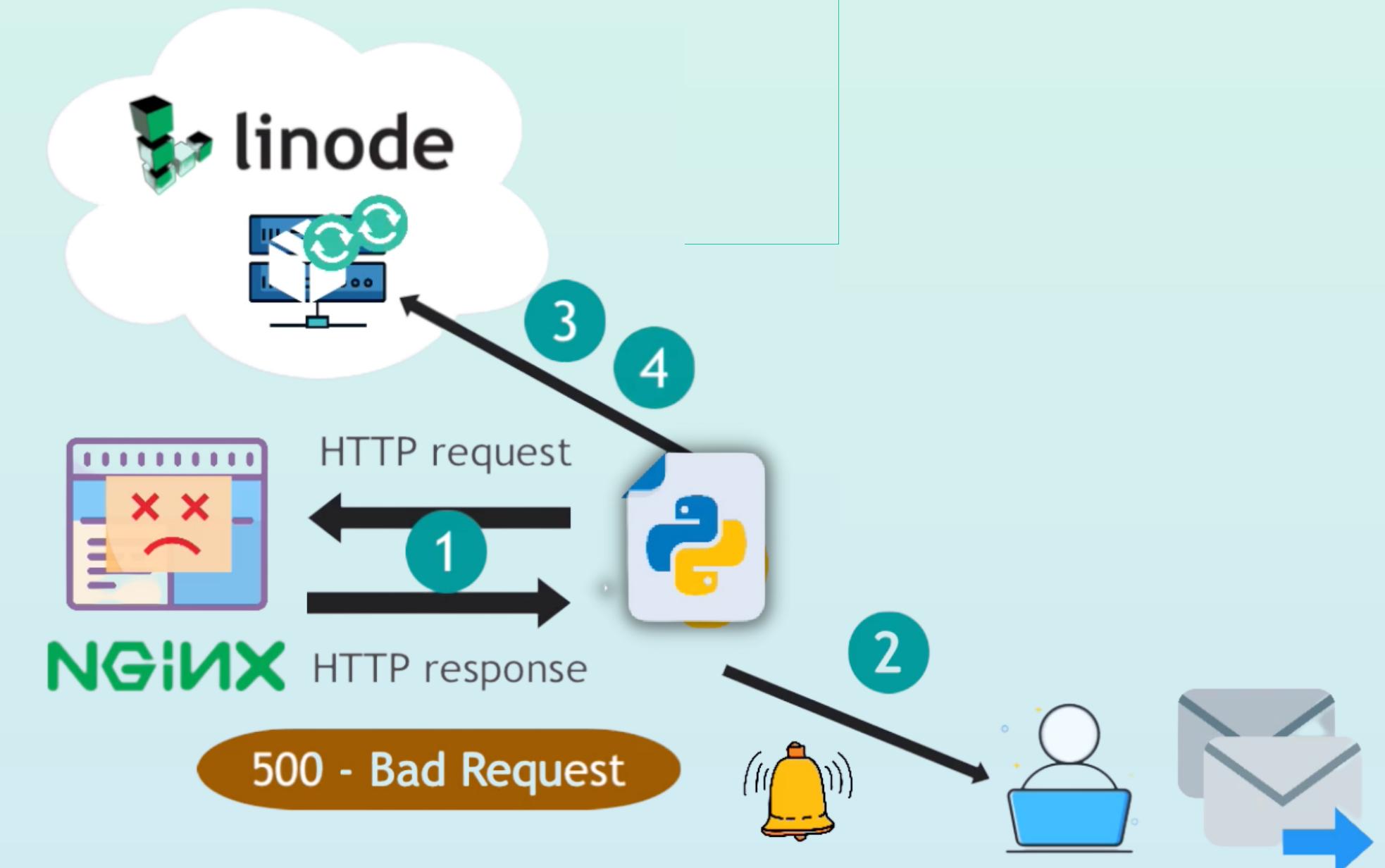
## Demo Project: Website Monitoring

### Write Automation Program for monitoring:

- 1) Write Python program that checks application
- 2) Send email when website is down

### Write Automation Program to fix the problem:

- 3) Re-start Docker container
- 4) Re-start server



# Handling Errors in Python Scripts - 1

- It's important to handle possible errors in your Python programs properly, because a lot can go wrong
- If errors are not handled correctly, it can lead to data loss, corrupt data etc.

- ✖ Snapshot creation failed
- ✖ Volume Attachment failed
- ✖ Volume Creation from Snapshot failed
- ✖ ...



Handle the error specific to the task

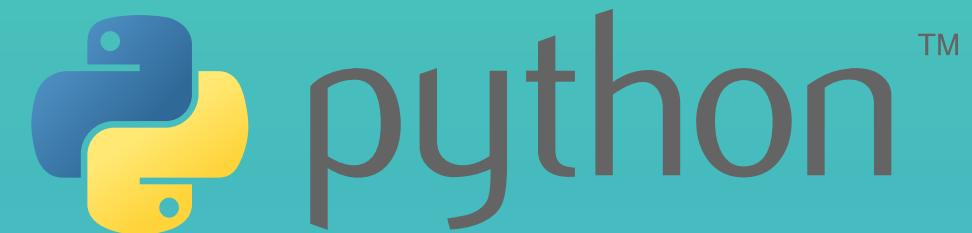


Log the error for debugging

## Snapshot Creation Failure:

- **Good solution:** Delete Snapshot and re-try
  - **Bad:** Keep it
- 
- Generally good practice to **completely roll back** and clean up half-created resources

# Handling Errors in Python Scripts - 2



Python doesn't have a state, so we need to take care of that ourselves



Terraform takes care of states automatically