### **Platform Engineering Python Exercise: Automating AWS Resource Provisioning**

#### **Scenario:**

You've just joined a new company as a junior DevOps engineer. One of your team's key responsibilities is provisioning AWS resources for various development projects, such as EC2 instances, S3 buckets, and Route53 DNS records.

While the DevOps team maintains strict standards for resource creation, developers currently rely on you to provision resources for them. However, you want to introduce **platform engineering** to automate this process, allowing developers to create resources themselves—within the guidelines defined by your team.

#### **Your Task:**

You’ll develop a Python-based **CLI (Command Line Interface)** tool that lets developers create, update, and manage AWS resources while ensuring compliance with the DevOps standards. This self-service tool will allow developers to request resources without needing direct DevOps intervention, but only within the boundaries you've set.

### **Phase 1: Core CLI Development**

You’ll start by creating a Python CLI tool that enables developers to:

1. **EC2 Instances:**
   * **Create:** Allow the creation of a new EC2 instance, with the option to choose between two of the smallest instance types (t3.nano and t4g.nano) and will not allow to create more than two running instances.
   * **AMI Choice:** Let developers select between the latest Ubuntu AMI or the latest Amazon Linux AMI.
   * **Manage Instances:** Enable starting and stopping of EC2 instances, but only if they were created via your CLI.
   * **List Instances:** Provide a list of all EC2 instances created through the CLI (exclude instances created by others).
2. **S3 Buckets:**
   * **Create:** Enable the creation of new S3 buckets, with a choice between public and private access.
   * **Confirmation for Public Buckets:** If public access is chosen, request additional approval (ask the user: "Are you sure?").
   * **File Upload:** Allow developers to upload files to an S3 bucket, but only if the bucket was created through the CLI.
   * **List Buckets:** Provide a list of all S3 buckets created by the CLI.
3. **Route53:**
   * **Create Zones:** Allow the creation of DNS zones via Route53.
   * **Manage DNS Records:** Enable developers to create, update, or delete DNS records, but only for zones created through the CLI.

#### **CLI Requirements:**

* The CLI should accept resource type, action (create, update, delete), and any required parameters.
* The CLI should provide clear output indicating the success or failure of the operation and the current status of the resource.

### **Phase 2: Bonus Challenge – Adding a UI**

For those looking for an extra challenge, integrate your Python CLI tool with an open-source tool like **Jenkins** to provide a user interface (UI). Each resource and action can be implemented as a different screen or job in Jenkins, providing a more accessible platform for developers to use.

### **Phase 3: Master DevOps Challenge – Creating a RESTful API**

For a more advanced project, wrap your CLI tool with a **RESTful API**. This will enable developers and other tools to interact with your platform programmatically, making resource creation and management even more scalable and flexible.

### **Summary:**

In this exercise, you will:

* Develop a Python CLI that simplifies AWS resource provisioning for developers while enforcing your standards.
* Use the Python package boto3 to manage the AWS resources, and use tags as a way to manage the resources you create. If you want to try and challenge yourself, you can try using the [Pulumi](https://www.pulumi.com/registry/packages/aws/) python packages instead or in addition to boto3.
* Provide self-service capabilities for creating, managing, and deleting AWS resources like EC2 instances, S3 buckets, and Route53 DNS records.
* Optionally extend the project with a UI and/or RESTful API to enhance usability and flexibility.

Good luck, and happy coding! 🎯