EXERCISE 1

Aim of the Project

The project focuses on the domain of cloud storage configuration across multiple cloud providers, including AWS , Azure, and Google Cloud Storage. The DSL will allow users to specify storage configurations such as provider, region, and access control in a concise, unified syntax. The scope includes automating the generation of provider-specific CLI or API commands and validating configurations, with plans to incorporate advanced features like lifecycle policies and cross-provider resource synchronization.

Objectives of the Project

The primary tasks to be automated include:

1. **Resource Definition:** Automating the setup of storage resources (e.g., buckets or containers) across providers.
2. **Region Specification:** Simplifying region selection for optimal latency and compliance.
3. **Cross-Provider Compatibility:** Ensuring a consistent experience for managing resources across multiple providers.

Examples and Potential Benefits

1. A company needs to set up an AWS S3 bucket for securely storing sensitive customer data. Using the DSL, the organization can quickly define the bucket name, region, and access control in a single step. This eliminates the need to manually write multiple commands or scripts specific to the cloud provider.

   Benefit: Reduces the complexity of configuration and minimizes the risk of errors, especially for non-technical users.

2. A user tries to configure a storage resource in a region not supported by a particular provider. The DSL immediately flags this issue with a clear error message, prompting the user to select a valid region.

   Benefit: Prevents deployment failures by validating configurations before execution, saving time and avoiding potential misconfigurations.

Methodology and Scope

This project focuses on creating a Domain-Specific Language (DSL) for automating cloud storage configurations across multiple providers like AWS, Azure, and Google Cloud. The DSL will enable users to define concise scripts to configure cloud storage resources, and the platform will generate the required configuration scripts or API calls automatically.

The methodology follows an **iterative and incremental approach**, focusing on problem analysis, design, implementation, and testing. The aim is to progressively refine the DSL, ensuring it covers essential features and aligns with real-world use cases.

Step 1: Problem Analysis and Requirements

The first step involves identifying key aspects of cloud storage configuration, such as naming, region selection, access control, and cross-provider compatibility. For instance, AWS S3, Azure Blob Storage, and Google Cloud Storage share similarities in these features, making it possible to design a unified abstraction. The DSL's syntax and semantics will be tailored to support these shared properties while allowing extensions for provider-specific features.

Step 2: DSL Design

The design phase focuses on creating the DSL's metamodel using class diagrams. Key concepts like `Storage`, `Provider`, `Region`, and `AccessControl` will be modeled, including their properties. The DSL syntax will be straightforward, allowing users to write scripts like:

```plaintext
storage "my_bucket" {
   provider = "AWS"
   region = "us-west-2"
   access = "private"
}
```

This script would define a private AWS S3 bucket in the `us-west-2` region.

Step 3: Implementation Phases

The project will be developed in three iterations:

1. Iteration 1: Basic Cloud Storage Setup

The first iteration implements the DSL for a single provider, such as AWS. It focuses on essential features like defining storage names, regions, and access levels. The output will be CLI scripts or JSON configurations.

2. Iteration 2: Cross-Provider Support

The second iteration extends the DSL to support multiple providers, including Azure and Google Cloud. Syntax and functionality will adapt to different providers while maintaining consistency.

3. Iteration 3: Advanced Features

The final iteration introduces advanced options like validation, lifecycle policies, and error handling. For example, the DSL could validate unsupported regions or flag missing mandatory parameters.

Step 4: Testing and Validation

Generated outputs will be tested against real-world scenarios. For instance, the DSL could create an AWS S3 bucket using:

```bash

aws s3api create-bucket --bucket my_bucket --region us-west-2

aws s3api put-bucket-acl --bucket my_bucket --acl private

```

Similarly, for Azure, the DSL might produce:

```bash

az storage container create --name logs_container --public-access container --region eastus

```