Provision a Cloud Service with IaC to Serve a Dynamic HTML Page

Project Summary

In this project, I provisioned a service on AWS using Terraform for Infrastructure as Code (IaC). The solution consists of an HTTP API exposed via API Gateway that invokes an AWS Lambda function. This Lambda function generates and serves a dynamic HTML page whose content is a string stored and managed in AWS Systems Manager Parameter Store.
This architecture allows modifying the dynamic content of the page without the need to redeploy or modify the infrastructure, since the Lambda function reads the updated value directly from Parameter Store on every execution.

Solution Description and Available Options

AWS Lambda + API Gateway: Used to build a serverless solution that provides a public HTTP endpoint. Lambda executes the logic to retrieve the dynamic string and construct the HTML.

AWS Systems Manager Parameter Store (SSM): The dynamic string value is stored here and read in real time by Lambda. This enables modifying content without redeployments.

Terraform: Tool to declare and provision the entire infrastructure as code, enabling version control, reproducibility, and automated deployment.

Other options considered

Amazon S3 + CloudFront: Suitable for static page hosting but does not support dynamic content without redeploy or backend logic.

EC2 servers or containers: Offer full flexibility but involve higher operational complexity and costs, plus scaling management.

Other serverless functions: Azure Functions or Google Cloud Functions could be used in other clouds, but AWS was chosen due to integration and familiarity.

Decisions Made and Justification

AWS Lambda was chosen due to its serverless nature, removing the need to manage servers or underlying infrastructure, allowing automatic scaling and cost efficiency.

API Gateway allows exposing Lambda via HTTP without configuring web servers or load balancers.

Parameter Store was used to store the dynamic value because of its ease of managing secure, versioned parameters and its native integration with Lambda.

Terraform ensures that infrastructure is reproducible, auditable, and easily modifiable, ideal for team projects and automated deployments.

Improvements and Extensions for the Future

Authentication and access control: Add mechanisms to restrict who can update the dynamic string or access the endpoint.

Admin UI panel: Develop a web interface for authorized users to modify the string without using CLI or console.

Monitoring and alerts: Integrate CloudWatch and SNS to monitor service health and send alerts on failures.

Automated tests and continuous integration (CI/CD): Implement pipelines to validate Terraform changes and Lambda code.

Caching: Add caching to reduce latency and lower Lambda invocation costs.
Support for multiple strings or versions: Expand to handle different dynamic messages depending on user or context.

Instructions to Run the Project

1. Configure AWS CLI: Install AWS CLI and configure a profile with appropriate credentials:

```
bash
aws configure --profile terraform-user
```

2. Deploy Infrastructure with Terraform: From the folder containing the Terraform code, run:

```
bash
terraform init
terraform apply
```

Confirm that API Gateway, Lambda, and necessary parameters are created.

3. Update the Dynamic String: To change the text shown on the page without redeploying:

```
bash
aws ssm put-parameter \
  --name "/dynamic/string" \
  --value "new-value" \
  --type String \
  --overwrite \
  --profile terraform-user \
  --region us-east-1
```

4. Test the Application: Get the public API Gateway URL (Terraform output):

```
bash
terraform output api_endpoint
```

5. Open in a browser or use curl:

```
bash
curl $(terraform output -raw api_endpoint)
```

The response will display the updated dynamic string.