Deploy AWS Lambda Functions and Layers using Terraform

Executive Summary

This document outlines the implementation of Infrastructure as Code (IaC) using Terraform to deploy AWS Lambda functions with custom Lambda Layers and S3 storage infrastructure. The solution follows modular design principles and implements security best practices including encryption and versioning.

Table of Contents

1. Introduction

Objective

The primary objective of this project is to design and implement a scalable, secure, and maintainable infrastructure for deploying AWS Lambda functions using Terraform. The implementation includes:

- Custom S3 bucket creation for Lambda artifact storage
- KMS encryption for data security
- Lambda Layers for code reusability
- Modular Terraform configuration for easy maintenance

Scope

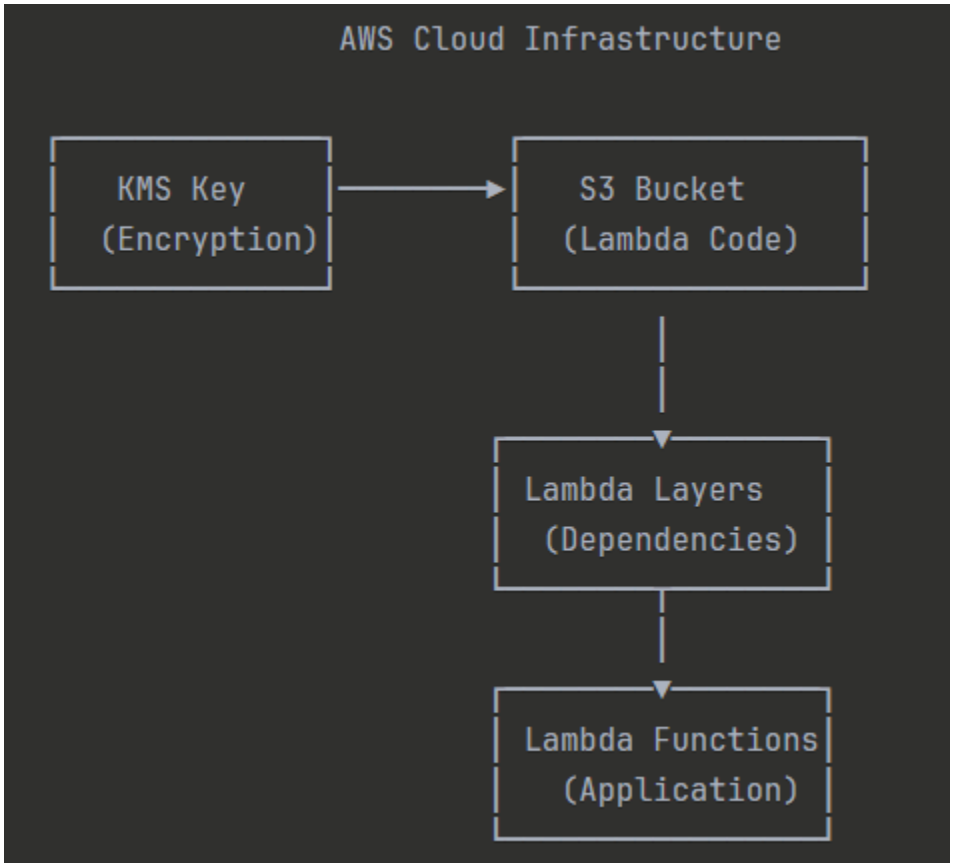
This implementation covers the following AWS services:

- **AWS Lambda**: Serverless compute functions
- **AWS Lambda Layers**: Shared dependencies and libraries
- **AWS S3**: Object storage for Lambda code artifacts
- **AWS KMS**: Encryption key management

2. Architecture Overview

High-Level Architecture

The architecture consists of the following components:



Component Relationships

1. **KMS Key** encrypts data stored in the S3 bucket
2. **S3 Bucket** stores Lambda deployment packages and layer artifacts
3. **Lambda Layers** provide reusable dependencies for Lambda functions
4. **Lambda Functions** utilize layers and are deployed from S3

3. Project Structure

Directory Layout

terraform-lambda-project/
```
│
├── main.tf                 # Main Terraform configuration
├── variables.tf            # Input variables
├── outputs.tf              # Output values
├── locals.tf               # Local values and data transformations
│
├── modules/
│   └── s3/                 # Custom S3 module
│       ├── main.tf
│       ├── variables.tf
│       └── outputs.tf
│
└── lambda/
    ├── function1/          # Lambda function 1 source code
    ├── function2/          # Lambda function 2 source code
    └── layers/             # Lambda layers source code
        ├── layer1/
        └── layer2/
```

4. Implementation Details

4.1 KMS Key Configuration

The implementation begins with creating a KMS (Key Management Service) key for encrypting S3 bucket data:

```
resource "aws_kms_key" "s3_bucket_key" {
  description             = "KMS key for S3 bucket encryption"
  deletion_window_in_days = 10

  tags = {
    Name = "${var.environment}-s3-bucket-key"
  }
}
```

Key Features:

- Provides server-side encryption for S3 objects
- 10-day deletion window for key recovery
- Environment-based tagging for resource management

4.2 S3 Bucket Module

The S3 bucket is created using a custom module that enables encryption and versioning:

```
module "lambda_s3_bucket" {
  source       = "./modules/s3"
  bucket_name  = "${var.environment}-lambda-bucket-307946636515"
  if_encrypted = true
  if_versioning = true
  kms_key_arn  = aws_kms_key.s3_bucket_key.arn
  log_bucket   = "${var.environment}-log-bucket-307946636515"
}
```

Configuration Details:

- **Encryption**: Enabled using the KMS key
- **Versioning**: Enabled for artifact history tracking
- **Logging**: Configured to send access logs to a separate log bucket
- **Naming Convention**: Includes environment prefix and AWS account ID

4.3 Lambda Layers

Lambda Layers are deployed using the community Terraform AWS Lambda module with a `for_each` loop for multiple layers:

```
module "lambda_layer" {
  for_each = local.lambda_layers_map
  source   = "terraform-aws-modules/lambda/aws"
  version = "8.1.0"

  create_layer = true

  layer_name          = each.key
  compatible_runtimes = each.value.compatible_runtimes
  runtime             = "python3.12"

  source_path = {
    path            = "${path.module}/${each.value.path}",
```

```
    pip_requirements = true,
    prefix_in_zip    = "python"
  }
  store_on_s3 = true
  s3_bucket   = module.lambda_s3_bucket.bucket_name
}
```

Layer Features:

- **Dynamic Creation**: Uses `for_each` to create multiple layers from a map
- **Python 3.12 Runtime**: Configured for the latest Python runtime
- **Pip Requirements**: Automatically handles Python package dependencies
- **S3 Storage**: Layer artifacts are stored in the created S3 bucket
- **Proper Packaging**: Uses `prefix_in_zip = "python"` for correct layer structure

4.4 Lambda Functions

Lambda functions are deployed with layers attached:

```
module "lambda_function" {
  for_each = local.lambda_info_map
  source   = "terraform-aws-modules/lambda/aws"
  version = "8.1.0"

  function_name = each.value.name
  handler       = each.value.handler
  runtime       = each.value.runtime
  publish       = true
  layers = [for layer_name in each.value.layers :
module.lambda_layer[layer_name].lambda_layer_arn ]
  timeout       = 60
  store_on_s3 = true
  s3_bucket   = module.lambda_s3_bucket.bucket_name

  source_path = each.value.path
  environment_variables = each.value.environments_variables

  tags = {
    repo = "august-bootcamp"
  }
}
```

Function Configuration:

- **Dynamic Deployment**: Multiple functions created from map variable
- **Layer Integration**: Dynamically attaches required layers using ARNs
- **Version Publishing**: Enabled for versioning support
- **Timeout**: Set to 60 seconds
- **Environment Variables**: Custom variables per function
- **S3 Storage**: Deployment packages stored in S3 for larger artifacts

5. Key Components

5.1 For_each Loops

The implementation uses `for_each` meta-argument to create multiple resources dynamically:

for_each = local.lambda_layers_map

This approach provides:

- Scalability: Easy to add/remove resources
- Maintainability: Single configuration block for multiple resources
- Resource addressing: Each resource has a unique identifier

5.2 Module References

Module outputs are referenced using the syntax:

module.lambda_s3_bucket.bucket_name
module.lambda_layer[layer_name].lambda_layer_arn

5.3 Dynamic Layer Attachment

Layers are dynamically attached to functions using list comprehension:

layers = [for layer_name in each.value.layers :
module.lambda_layer[layer_name].lambda_layer_arn]

This creates a list of layer ARNs based on the layer names specified in the function configuration.

6. Security Features

6.1 Encryption at Rest

- **KMS Encryption**: All S3 objects are encrypted using customer-managed KMS keys
- **Key Rotation**: Can be enabled on the KMS key for enhanced security

6.2 Versioning

- **S3 Versioning**: Enabled to maintain history of Lambda artifacts
- **Recovery**: Ability to rollback to previous versions if needed

6.3 Access Logging

- **S3 Access Logs**: Configured to track bucket access patterns
- **Audit Trail**: Maintains compliance and security monitoring

6.4 Best Practices Implemented

1. Separate log bucket for access logs
2. Environment-based resource naming
3. Tagging strategy for resource management
4. Modular design for reusability

7. Deployment Process

Prerequisites

1. AWS Account with appropriate permissions
2. Terraform installed (version 1.0+)
3. AWS CLI configured with credentials
4. Lambda function source code and layer dependencies prepared

Deployment Steps

Step 1: Initialize Terraform
terraform init

This command:

- Downloads required provider plugins
- Initializes the backend

- Downloads referenced modules

Step 2: Review Execution Plan
terraform plan

This command:

- Shows resources to be created/modified/destroyed
- Validates configuration syntax
- Displays estimated changes

Step 3: Apply Configuration
terraform apply

This command:

- Creates all defined resources
- Uploads Lambda code to S3
- Deploys Lambda functions and layers

Step 4: Verify Deployment
terraform output
aws lambda list-functions
aws s3 ls

Resource Creation Order

Terraform automatically manages dependencies and creates resources in the correct order:

1. KMS Key
2. S3 Bucket (depends on KMS key)
3. Lambda Layers (depends on S3 bucket)
4. Lambda Functions (depends on layers and S3 bucket)

8. Code Explanation

Variables Structure

The implementation uses local variables to define Lambda configurations:

lambda_layers_map - Defines layer configurations:

```
local.lambda_layers_map = {
  "layer1" = {
    compatible_runtimes = ["python3.12"]
    path = "lambda/layers/layer1"
  }
  "layer2" = {
    compatible_runtimes = ["python3.12"]
    path = "lambda/layers/layer2"
  }
}
```

lambda_info_map - Defines function configurations:

```
local.lambda_info_map = {
  "function1" = {
    name = "my-function-1"
    handler = "index.handler"
    runtime = "python3.12"
    path = "lambda/function1"
    layers = ["layer1", "layer2"]
    environments_variables = {
      ENV = "production"
    }
  }
}
```

Module Usage Pattern

The implementation follows a consistent pattern:

1. **Source Module**: Uses official AWS Lambda Terraform module
2. **Version Pinning**: Specifies version 8.1.0 for consistency
3. **Dynamic Configuration**: Uses map variables for scalability
4. **Output References**: Links resources using module outputs

9. Benefits and Best Practices

Benefits of This Implementation

1. **Infrastructure as Code**
    - Version-controlled infrastructure

    - Reproducible deployments
    - Documentation through code
2. **Modularity**
    - Reusable S3 module
    - Standardized Lambda deployment
    - Easy to extend and modify
3. **Scalability**
    - Add functions/layers by updating variables
    - No code duplication
    - Consistent configuration
4. **Security**
    - Encryption at rest
    - Access logging
    - Least privilege access (can be extended with IAM roles)
5. **Cost Optimization**
    - Shared dependencies via layers
    - S3 storage for large deployment packages
    - Efficient resource utilization

Best Practices Followed

1. **Module Version Pinning**: Ensures consistent behavior across deployments
2. **Resource Tagging**: Enables cost tracking and resource management
3. **Environment Naming**: Separates resources by environment
4. **Encryption**: Protects sensitive data
5. **Versioning**: Enables rollback capabilities
6. **For_each Usage**: Better than count for resource creation
7. **Output References**: Proper dependency management between resources

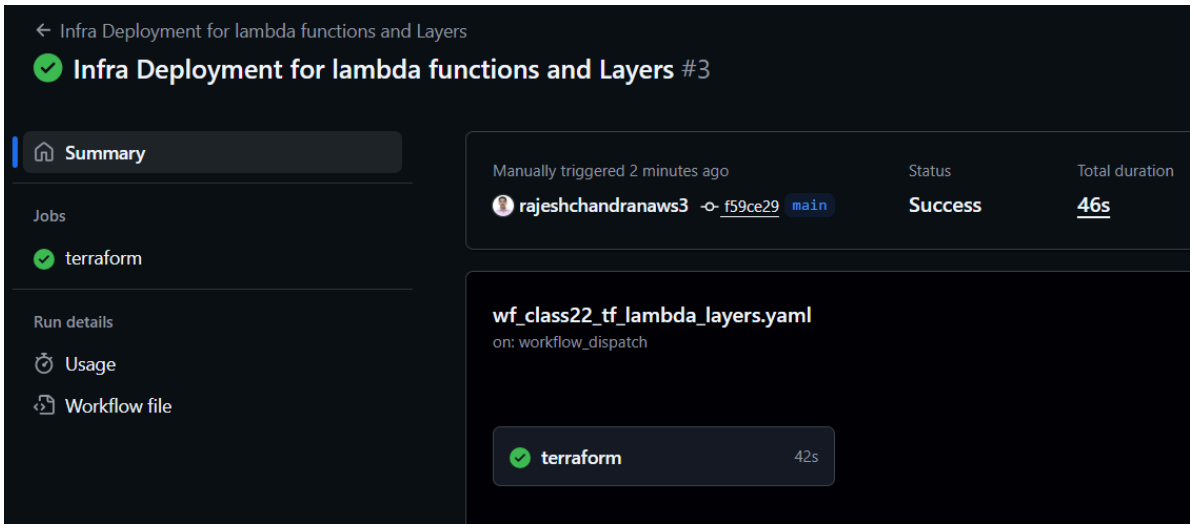
10. GitHub Actions

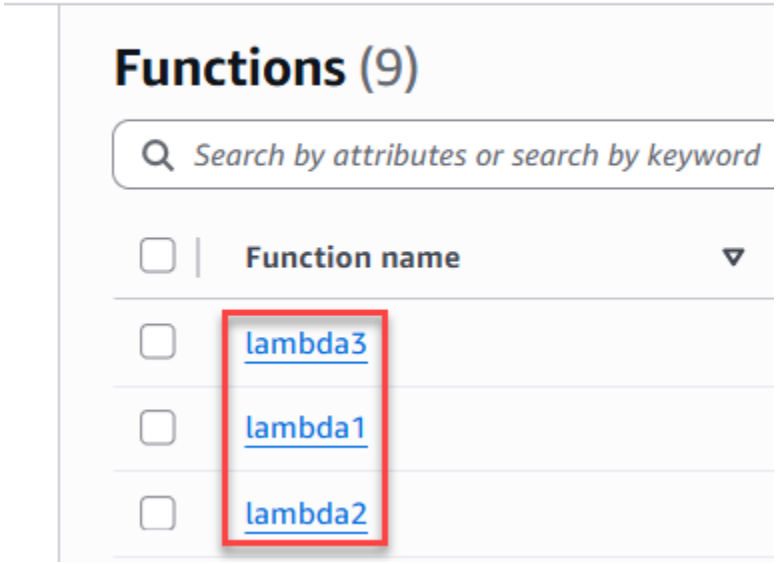
Implemented for manual trigger, using workflow_dispatch.

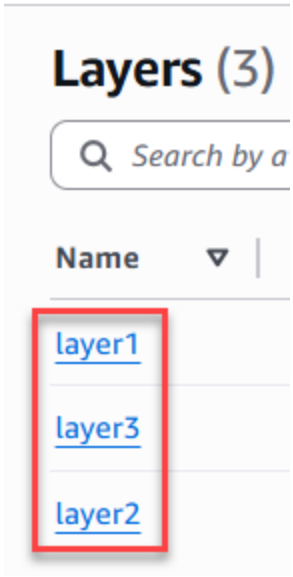
```yaml
name: Infra Deployment for lambda functions and Layers

on:
  workflow_dispatch:
    inputs:
      terraform_action:
        description: 'apply/destroy'
```

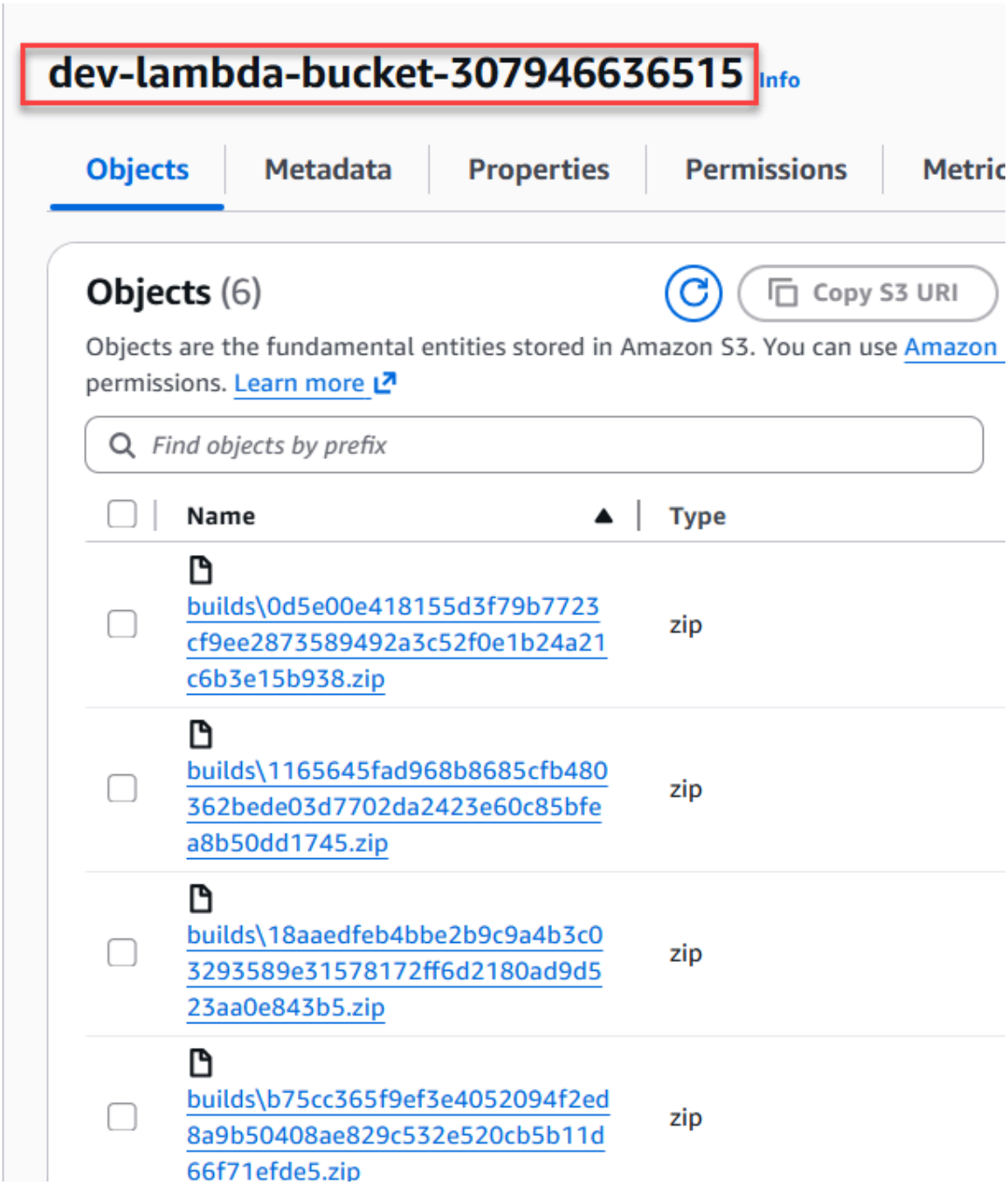
```
        required: true

env:
  terraform_action: ${{ github.event.inputs.terraform_action }}
  terraform_version: 1.8.1
  working_directory:
class22-lambda-layers-terraform/real-world-code/production-way/infra
  AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
  AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
  AWS_REGION: us-east-1

jobs:
  terraform:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Install Terraform
        uses: hashicorp/setup-terraform@v2
        with:
          terraform_version: ${{ env.terraform_version }}

      - name: Configure AWS Credentials
        uses: aws-actions/configure-aws-credentials@v5
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-1

      - name: Terraform Init
        working-directory: ${{ env.working_directory }}
        run: terraform init

      - name: Terraform Apply
        working-directory: ${{ env.working_directory }}
        run: terraform ${{ env.terraform_action }} -auto-approve
```

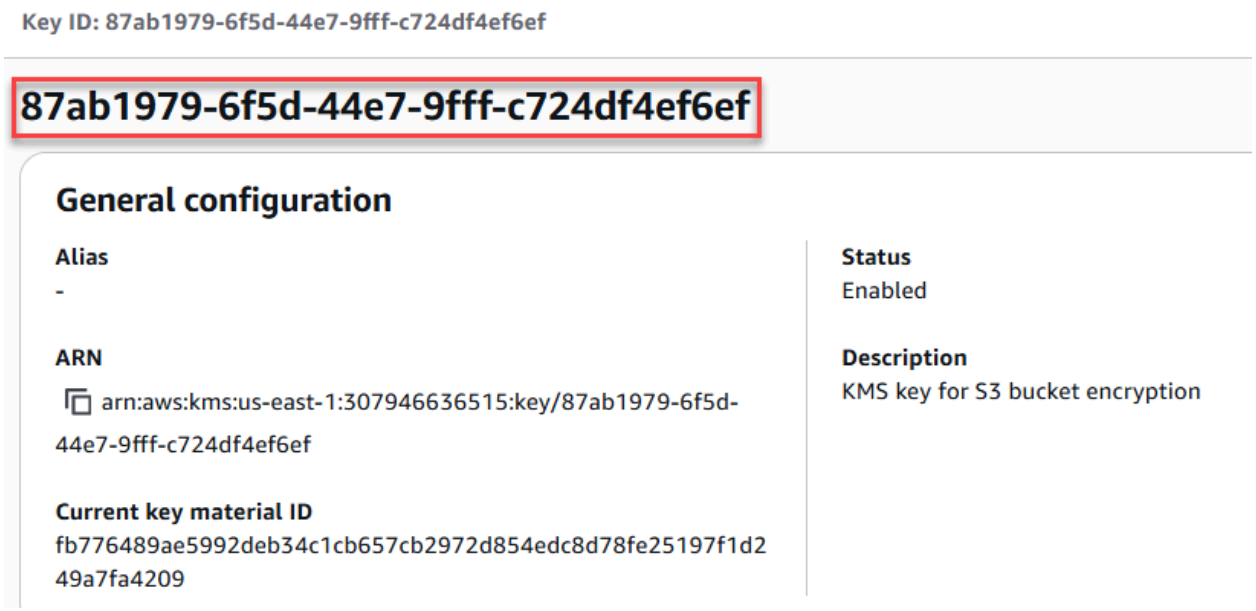
11. Output


Infra Deployment for lambda functions and Layers
Infra Deployment for lambda functions and Layers #3
Summary
Jobs
terraform
Run details
Usage
Workflow file
Manually triggered 2 minutes ago
rajeshchandranaws3
f59ce29
main
Status
Success
Total duration
46s
wf_class22_tf_lambda_layers.yaml
on: workflow_dispatch
terraform
42s


Functions (9)
Search by attributes or search by keyword
Function name
lambda3
lambda1
lambda2


Layers (3)
Search by a
Name
layer1
layer3
layer2


dev-lambda-bucket-307946636515
Info
Objects
Metadata
Properties
Permissions
Metric
Objects (6)
Copy S3 URI
Objects are the fundamental entities stored in Amazon S3. You can use Amazon
permissions. Learn more
Find objects by prefix
Name
Type
builds\0d5e00e418155d3f79b7723cf9ee2873589492a3c52f0e1b24a21c6b3e15b938.zip
zip
builds\1165645fad968b8685cfb480362bede03d7702da2423e60c85bfea8b50dd1745.zip
zip
builds\18aaedfeb4bbe2b9c9a4b3c03293589e31578172ff6d2180ad9d523aa0e843b5.zip
zip
builds\b75cc365f9ef3e4052094f2ed8a9b50408ae829c532e520cb5b11d66f71efde5.zip
zip



12. Conclusion

This Terraform implementation demonstrates a robust, scalable, and secure approach to deploying AWS Lambda functions with layers and S3 storage. The modular design allows for easy maintenance and extension, while security features like KMS encryption and S3 versioning ensure data protection and recovery capabilities.

Key Achievements

✓ Successfully implemented Infrastructure as Code using Terraform
✓ Created reusable module for S3 bucket configuration
✓ Deployed Lambda layers for shared dependencies
✓ Configured Lambda functions with dynamic layer attachment
✓ Implemented encryption and versioning for security
✓ Followed AWS and Terraform best practices

Learning Outcomes

Through this project, the following concepts were successfully demonstrated:

- Terraform module development and usage
- AWS Lambda deployment patterns
- S3 bucket configuration and security
- KMS encryption implementation
- Infrastructure as Code best practices
- Dynamic resource creation using for_each

References

- Terraform AWS Lambda Module Documentation
- AWS Lambda Developer Guide
- Terraform Documentation
- AWS KMS Documentation
- AWS S3 Documentation