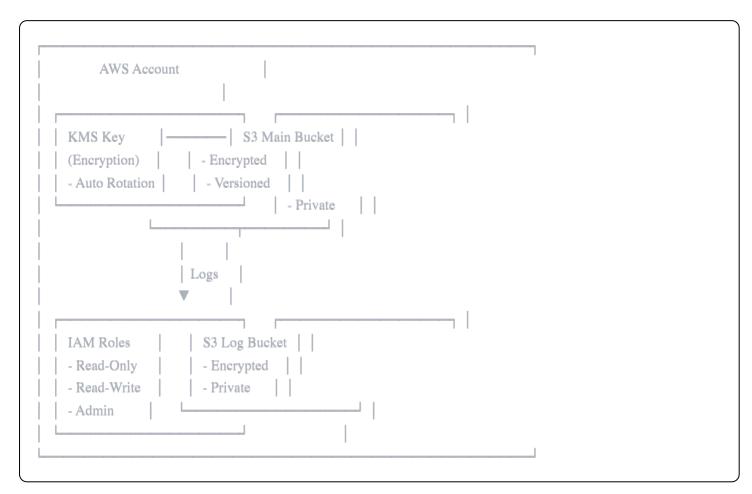
Secure S3 Bucket Infrastructure with Terraform

A production-ready, security-hardened S3 infrastructure built with Terraform, demonstrating cloud security best practices and the principle of least privilege.

Overview

This project implements a secure S3 storage solution with enterprise-grade encryption, access controls, and audit logging. All infrastructure is defined as code using Terraform, making it repeatable, testable, and version-controlled.

Architecture



Security Features

Encryption

- KMS Customer-Managed Keys: Full control over encryption keys
- Automatic Key Rotation: Annual rotation enabled for compliance
- S3 Bucket Keys: 99% reduction in KMS request costs
- Encryption at Rest: All data encrypted using AES-256 via KMS

Access Controls

• Public Access Blocked: All four public access settings enabled

• Least Privilege IAM Roles: Three distinct roles with minimal permissions

• Read-Only: List and read objects only

• Read-Write: Upload and read, but cannot delete

• Admin: Full access (tightly controlled)

• Instance Profiles: Secure role assumption for EC2 instances

Audit & Compliance

• Access Logging: All bucket access logged to separate log bucket

• Versioning: Object versions preserved for recovery and compliance

• Lifecycle Policies: Automated data retention and cost optimization

• Separation of Concerns: Logs stored in dedicated bucket

Data Protection

• Versioning: Protects against accidental deletion or malicious changes

• Lifecycle Management:

• Old versions deleted after 90 days

• Data transitions to cheaper storage classes (IA after 30 days, Glacier after 90 days)

Infrastructure Components

Resource Type	Count	Purpose
S3 Buckets	2	Main data bucket + dedicated log bucket
KMS Keys	1	Customer-managed encryption key
IAM Roles	3	Least-privilege access control
IAM Policies	3	Fine-grained permissions
Instance Profiles	3	EC2 role assumption

Total Resources Managed: 14

Prerequisites

- AWS Account with appropriate permissions
- AWS CLI configured with credentials
- Terraform >= 1.0
- Basic understanding of AWS IAM and S3

Deployment Instructions

1. Clone and Configure

```
bash

# Navigate to project directory

cd secure-s3-terraform

# Initialize Terraform

terraform init
```

2. Review the Plan

```
bash
# Preview what will be created
terraform plan
```

3. Deploy Infrastructure

```
bash

# Create all resources
terraform apply

# Type 'yes' when prompted
```

4. Verify Deployment

```
# List your buckets
aws s3 ls

# Check encryption settings
aws s3api get-bucket-encryption --bucket <your-bucket-name>

# List IAM roles created
aws iam list-roles --query 'Roles[?contains(RoleName, `s3-bucket`)].RoleName'
```

Outputs

After deployment, Terraform displays:

Usage Examples

Assigning Roles to EC2 Instances

```
# Launch EC2 with read-only access
aws ec2 run-instances \
--iam-instance-profile Name=s3-read-only-instance-profile \
--image-id ami-xxxxx \
--instance-type t2.micro
```

Testing Permissions

```
# From an EC2 with read-only role

aws s3 ls s3://my-secure-bucket-xxxxxxxx # Works

aws s3 cp file.txt s3://my-secure-bucket-xxxxxxxx/ # X Access Denied
```

Cost Estimation

Service	Monthly Cost (estimate)
KMS Key	~\$1.00
S3 Storage (first 50GB)	~\$1.15
S3 Requests	~\$0.01
Total	~\$2.16/month

Costs vary based on actual usage and data stored

Security Best Practices Implemented

- Encryption in Transit and at Rest: All data encrypted
- ✓ Principle of Least Privilege: Minimal permissions per role
- **Defense in Depth**: Multiple security layers

- ✓ Audit Logging: Complete access trail
- Separation of Duties: Different roles for different needs
- ✓ Infrastructure as Code: Repeatable, testable configurations
- **Wey Rotation**: Automated cryptographic key rotation
- **✓ Data Resilience**: Versioning protects against data loss

Cleanup

To destroy all resources and avoid AWS charges:

```
bash

# Preview what will be deleted
terraform destroy --dry-run

# Delete all resources
terraform destroy

# Type 'yes' when prompted
```

Warning: This permanently deletes all buckets and their contents. Ensure you have backups if needed.

Project Structure

What I Learned

Through this project, I gained hands-on experience with:

- Infrastructure as Code: Writing production-ready Terraform configurations
- AWS Security Services: KMS, IAM, S3 security features
- Access Control: Implementing least-privilege IAM policies
- Encryption: Customer-managed keys vs AWS-managed keys
- Compliance Requirements: Logging, versioning, and audit trails
- Cost Optimization: S3 lifecycle policies and storage classes
- Security Best Practices: Defense in depth, separation of concerns

Future Enhancements

Potential improvements for this project:
Add S3 bucket policies with IP restrictions
■ Implement cross-region replication for disaster recovery
■ Add AWS Config rules for compliance monitoring
Set up CloudWatch alarms for security events
☐ Integrate with AWS Organizations for multi-account deployment
Add automated security scanning in CI/CD pipeline
■ Implement S3 Object Lock for regulatory compliance
Technologies Used
• Terraform - Infrastructure as Code
• AWS S3 - Object storage
AWS KMS - Key management and encryption
AWS IAM - Identity and access management
AWS CLI - Command line tools
Author
Built as a learning project to demonstrate cloud security engineering skills and infrastructure as code best practices.
License
This project is open source and available for educational purposes.

Note: This is a learning/portfolio project. For production use, additional security measures should be implemented based on your specific compliance and security requirements.