Final Group Project - Cloud Security Hardening Plan

<u>ASSESSMENT REPORT</u>

<u>GROUP-6</u>

Atharva Bhingarkar– 120239366

Zhiwen Zhu – 121325471

Abhijeet Kumar – 121376701

Table of Findings

Executive Summary & Focus Area Selection

This Cloud Security initiative directly addresses the most critical weaknesses identified during our midterm assessment of an AWS-hosted e-commerce platform. Based on findings that revealed unencrypted EBS volumes and RDS instances, exposed database credentials, hard-coded secrets, and overly permissive security group rules, we focused on two strategic domains: **Data Protection & Encryption** and **Network & Access Control**.

In the **Data Protection & Encryption** domain, we implemented AWS Key Management Service (KMS)-based encryption across all EBS volumes, RDS MariaDB instances, and S3 buckets to ensure data confidentiality at rest. We also eliminated hard-coded secrets by integrating AWS Secrets Manager with auto-generated credentials, governed under tightly scoped IAM policies. These measures closed key gaps in data-at-rest protection and established centralized, auditable key management.

For **Network and Access Control**, we re-architected the cloud environment using segmented VPCs with dedicated public and private subnets. RDS was moved into private subnets and made non-publicly accessible. EC2 access was hardened by limiting ingress to essential ports, and the architecture is designed to support SSH access through a bastion host, enforcing security boundaries. Security groups were refined to allow inter-service traffic on a least-privilege basis, and egress flows. RDS access was explicitly scoped to trusted application-tier resources only.

Collectively, these enhancements significantly reduced the platform’s attack surface, improved security posture, and aligned the infrastructure with AWS best practices and the NIST Cybersecurity Framework. The result is a resilient, auditable, and secure cloud environment capable of protecting sensitive customer and transactional data at scale.

Background: Connection to Midterm

2.1 Summary of Vulnerabilities:

Network & Access Control Security:

The identified vulnerabilities that relate to this specific focus area are as follows:

- Unused Private Subnet: Indicates a lack of proper network segmentation, potentially allowing lateral movement of threats.
- Overly Permissive Security Group Rules: Allows excessive access, increasing vulnerability to attacks.

Data Protection & Encryption:

The identified vulnerabilities that relate to this specific focus area are as follows:

- EBS Volume Not Encrypted: Risks exposure of data at rest.
- MariaDB Database Not Encrypted with Weak Password: Weakens data protection, increasing risk of unauthorized access.
- Lack of Key Management and secure backups: Insufficient detail on comprehensive key management practices and absence of secure backup configurations.

2.2 Improvements:

Network & Access Control Security

The improvements are as follows:

- **Improvement 1: Harden security group for EC2 instance:** Implementation of more restrictive security group limits access only to the necessary IP ranges and ports, thus fixing the overly permissive security group rules flaw.

- **Improvement 2: Harden Security group for RDS (only EC2 instance can access with SQL port):** In a similar vein as the earlier improvement, this ensures that only the authorized ec2 instance in the right VPC is able to access the RDS.This works towards fixing the lack of network access lists.

- **Improvement 3: Configure Bucket Policy(only allow ec2 instance to assume IAM role to access):** Implementation of strict policies so that only the authorized ec2 instance with the right IAM role can access the given S3 bucket. This also works towards fixing the lack of network access lists.

Data Protection & Encryption

The improvements are as follows:

- **Improvement 1: Instead of MariaDB on EC2 instance, use RDS:** This change addresses vulnerabilities that are associated with managing database security and encryption by leveraging AWS RDS features, thus fixing the issue of MariaDB being unencrypted.

- **Improvement 2: Encryption for S3, RDS, and EBS:** This improvement focuses on the lack of encryption throughout the EC2 instance. By encrypting the S3 bucket, the RDS and the EBS backups, this resolves the issue of the EBS volumes not being encrypted as well as lack of secure backups to restore from.

- **Improvement 3: Use AWS KMS for Key management (Encryption key):** Implementation of AWS KMS ensures secure management and rotation of encryption keys, which is vital for the implementation of both of the aforementioned improvements, and also solves the issue of lack of key management.

Objective

3.1 Focus Area 1: Data Protection and Encryption

After conducting a security assessment of this e-commerce cloud infrastructure, we found that none of its sensitive business data was classified or encrypted. Given that the data involved in its business may include customer information, transaction records, or system configurations, protecting data at rest is critical. This helps maintain data confidentiality, ensure compliance with regulatory requirements, and minimize the impact of unauthorized access or data breaches.

To address this issue, we have focused on improving the security posture of data storage services by implementing encryption and centralized key management. Our enhancements target the following AWS services:

- **Amazon S3**
- **Amazon RDS**
- **Amazon EBS**
- **AWS KMS**
- **AWS Secrets Manager**

Specific risks addressed by each improvement include:

1. **Migrating MariaDB from EC2 to Amazon RDS:**
   Ensures the isolation of core data, effectively reducing the risk of database data leakage or malicious deletion caused by EC2 instance breaches.

2. **Enable encryption for S3, RDS, and EBS:**
   Prevent data leaks caused by RDS snapshot leaks, stolen volumes, or unencrypted S3 buckets.

3. **Use AWS KMS for key management:**
   Implement centralized management of keys for multiple encryption services (such as S3, RDS, and EBS) along with IAM policies to achieve fine-grained permission control and prevent key exposure or unauthorized use.

4. **Store credentials in AWS Secrets Manager:**
   Avoided hard-coding RDS login credentials and keys in configuration files or environment variables. By using the automatic generation and rotation features of Secrets Manager, we can reduce the risk of source code leakage or credential misuse.

3.2 Focus Area 2: Network and Access Control Security

Based on our previous security assessment, we found significant deficiencies in the network configuration of this e-commerce cloud platform. These include a lack of network segmentation, resource access restrictions, and access control. With these misconfigurations, the attack surface of the cloud resources has been significantly expanded, making network access control a critical task for current security hardening.

To achieve this goal, we optimized the key components of the AWS network architecture, including:

- **VPC design (separation of public and private subnets)**
- **Security groups and network access control lists (NACLs)**
- **IAM roles for identity-based access control**
- **S3 bucket policies**

The specific risks addressed by each improvement include:

1. **Enhance EC2 instance security groups**

   By allowing SSH access only from trusted IP addresses and closing all unnecessary ports, you can reduce the risk of external attacks (such as brute force attacks or scans).

2. **Restrict RDS access to EC2 only:**

By tightly binding the database security group to the EC2 security group on port 3306, you eliminate the risk of the database being accessed by untrusted sources.

3. **Configure S3 bucket policies:**
   Reduce the risk of data leaks or unauthorized public access to S3 objects by implementing policy-based access controls.

Implementation Plan

4.1 Focus Area 1: Data Protection and Encryption

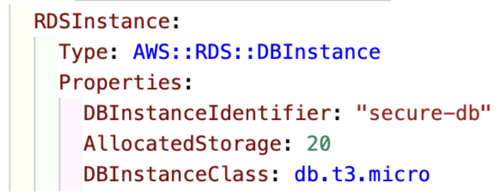
Improvement 1: Use RDS instead of MariaDB on EC2 instances.

Purpose: Replace self-managed MariaDB running on EC2 with Amazon RDS to improve data isolation, encryption support, backup automation, and reduce the attack surface.

Tools/AWS Services: RDS, CloudFormation Template

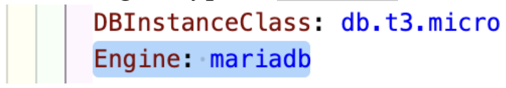
Technical Steps:

- Used AWS::RDS::DBinstance in CloudFormation



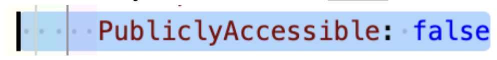
```
RDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBInstanceIdentifier: "secure-db"
    AllocatedStorage: 20
    DBInstanceClass: db.t3.micro
```

- Set the engine type to MariaDB



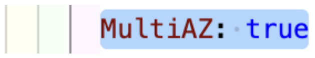
```
    DBInstanceClass: db.t3.micro
    Engine: mariadb
```

- Set PubiclyAccessible to false



```
    PubliclyAccessible: false
```

- Enable MultiAZ feature



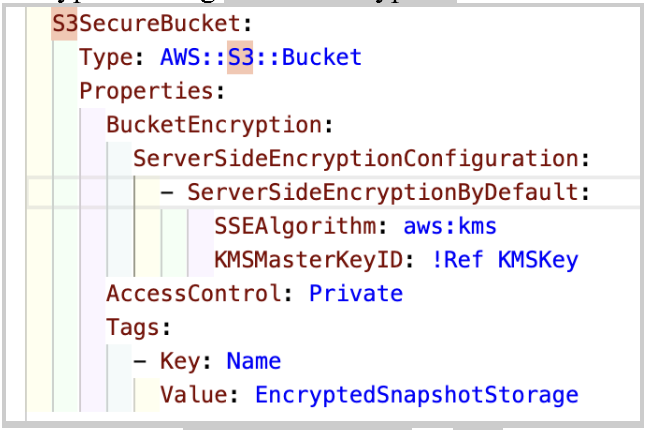
```
    MultiAZ: true
```

Improvement 2: Encryption for S3, RDS, and EBS

Purpose: By enabling encryption on all major storage resources, protect data at rest and minimize the risk of data breaches caused by stolen snapshots, S3 bucket configuration errors, or stolen EBS volumes.

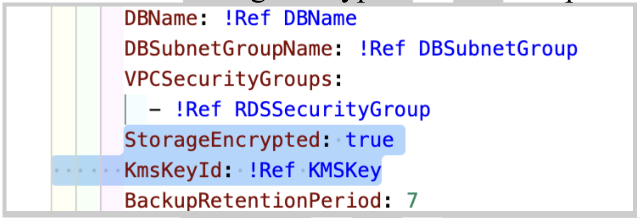
Tools/AWS Services: RDS, S3, EC2(EBS), and CloudFormation

Technical Steps:
- For S3: Server-side encryption is enabled by default. Enable KMS-based default encryption using BucketEncryption



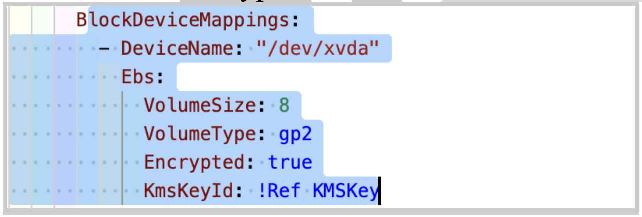
```
S3SecureBucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketEncryption:
      ServerSideEncryptionConfiguration:
        - ServerSideEncryptionByDefault:
            SSEAlgorithm: aws:kms
            KMSMasterKeyID: !Ref KMSKey
    AccessControl: Private
    Tags:
      - Key: Name
        Value: EncryptedSnapshotStorage
```

- For RDS: Set StorageEncrypted to true and provide the KmsKeyId



```
      DBName: !Ref DBName
      DBSubnetGroupName: !Ref DBSubnetGroup
      VPCSecurityGroups:
        - !Ref RDSSecurityGroup
      StorageEncrypted: true
      KmsKeyId: !Ref KMSKey
      BackupRetentionPeriod: 7
```

- For EBS: Set Encrypted to true in BlockDeviceMapping and associate the KmsKeyId



```
    BlockDeviceMappings:
      - DeviceName: "/dev/xvda"
        Ebs:
          VolumeSize: 8
          VolumeType: gp2
          Encrypted: true
          KmsKeyId: !Ref KMSKey
```

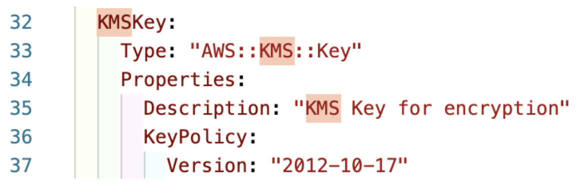
- All services use a unified KMS key created by AWS KMS

Improvement 3: Use AWS KMS for Key management (Encryption key)

Purpose: Centralized management of encryption keys to ensure security, support fine-grained IAM policies, logging, and key rotation.

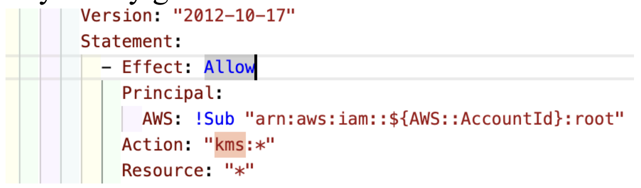
Tools/AWS Services: AWS KMS, CloudFormation

Technical Steps:
- Created a KMS CMK in CloudFormation



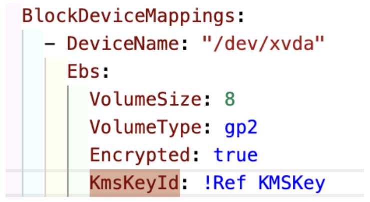
```
KMSKey:
  Type: "AWS::KMS::Key"
  Properties:
    Description: "KMS Key for encryption"
    KeyPolicy:
      Version: "2012-10-17"
```

- Key Policy grants root user full access

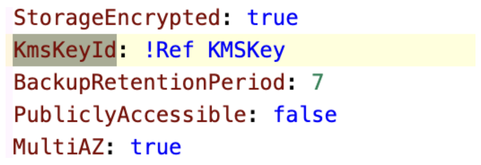


```
Version: "2012-10-17"
Statement:
  - Effect: Allow
    Principal:
      AWS: !Sub "arn:aws:iam::${AWS::AccountId}:root"
    Action: "kms:*"
    Resource: "*"
```

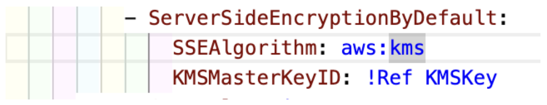
- RDS, EBS, and S3 all reference KMS keys via KmsKeyId to enable encryption.



```
BlockDeviceMappings:
  - DeviceName: "/dev/xvda"
    Ebs:
      VolumeSize: 8
      VolumeType: gp2
      Encrypted: true
      KmsKeyId: !Ref KMSKey
```



```
StorageEncrypted: true
KmsKeyId: !Ref KMSKey
BackupRetentionPeriod: 7
PubliclyAccessible: false
MultiAZ: true
```



```
- ServerSideEncryptionByDefault:
    SSEAlgorithm: aws:kms
    KMSMasterKeyID: !Ref KMSKey
```

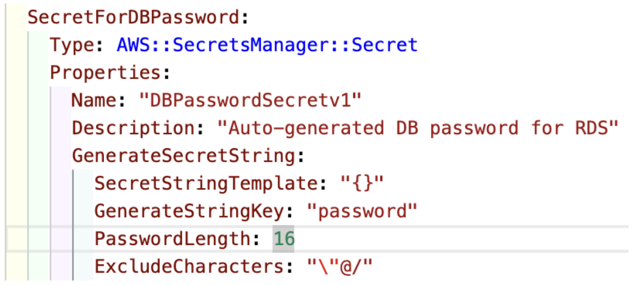
Improvement 4: Secure RDS key with Secrets Manager

Purpose: By securely storing database credentials in AWS Secrets Manager and enabling automatic generation and encryption, you eliminate the need for hard-coded or manually managed database credentials.

Tools/AWS Services: SecretsManager, RDS, AWS KMS, CloudFormation

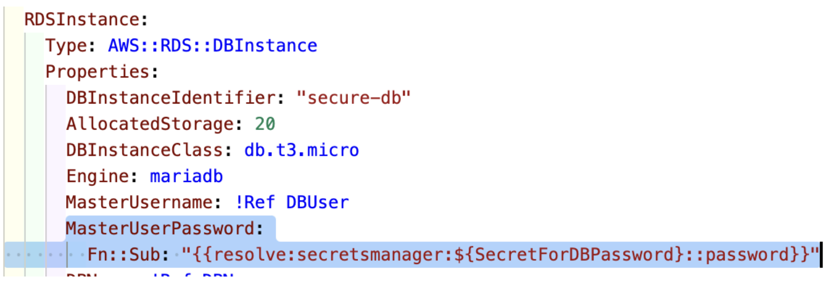
Technical Steps:

- Created a secret using SecretsManager with GenerateSecretString



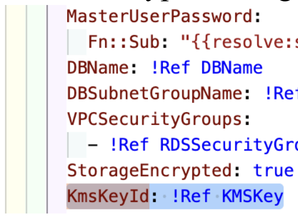
```
SecretForDBPassword:
  Type: AWS::SecretsManager::Secret
  Properties:
    Name: "DBPasswordSecretv1"
    Description: "Auto-generated DB password for RDS"
    GenerateSecretString:
      SecretStringTemplate: "{}"
      GenerateStringKey: "password"
      PasswordLength: 16
      ExcludeCharacters: "\"@/"
```

- Bound the secret to RDS by referencing it in MasterUserPassword



```
RDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBInstanceIdentifier: "secure-db"
    AllocatedStorage: 20
    DBInstanceClass: db.t3.micro
    Engine: mariadb
    MasterUsername: !Ref DBUser
    MasterUserPassword:
      Fn::Sub: "{{resolve:secretsmanager:${SecretForDBPassword}::password}}"
```

- Secret is encrypted using the same KMS key



```
    MasterUserPassword:
      Fn::Sub: "{{resolve:secretsmanager:${SecretForDBPassword}::password}}"
    DBName: !Ref DBName
    DBSubnetGroupName: !Ref DBSubnetGroup
    VPCSecurityGroups:
      - !Ref RDSSecurityGroup
    StorageEncrypted: true
    KmsKeyId: !Ref KMSKey
```

4.2 Focus Area 2: Network and Access Control Security

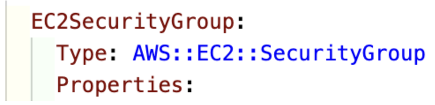
Improvement 1: Harden security group for EC2 instance

Purpose: Restrict inbound traffic to only necessary ports and trusted IP addresses to reduce the attack surface of EC2 instances.

Tools/AWS Services: EC2, Security Group, CloudFormation

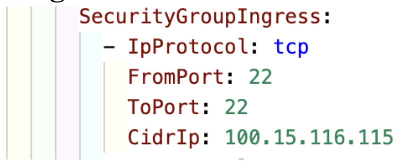
Technical Steps:

- EC2 security groups defined in CloudFormation



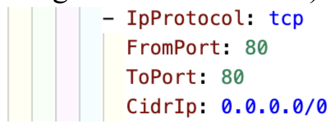
```
EC2SecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
```

- Allow inbound SSH connections only from my home IP address (100.15.116.115). **Note: Using this IP address in CloudFormation will generate an error with a warning!**



```
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 100.15.116.115
```

- Allow inbound HTTP connections from the public internet (0.0.0.0/0) for web access (Ideally, HTTPs should be applied, but we are not implementing this security configuration at this time)



```
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: 0.0.0.0/0
```

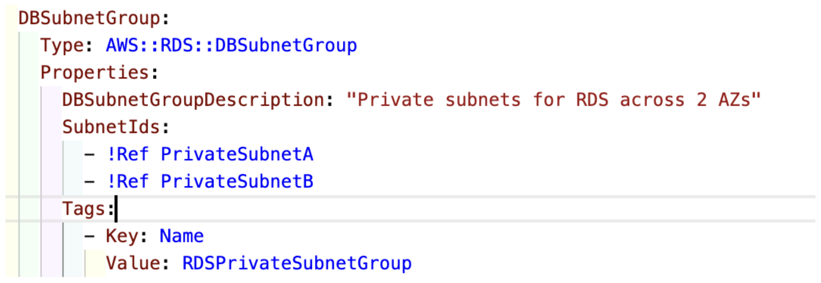
Improvement 2: Harden Security group for RDS (only EC2 instance can access with SQL port)

Purpose: Restrict database access to EC2 instances within the same VPC to ensure that RDS is isolated from external or unauthorized access.

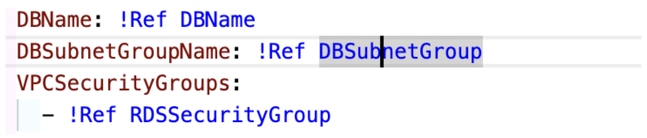
Tools/AWS Services: RDS, Security Group, Subnet, CloudFormation

Technical Steps:

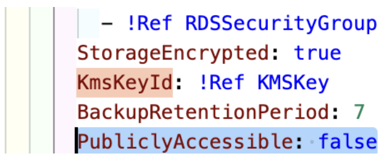
- Launch an RDS instance in a private subnet, set PubliclyAccessible to false, and avoid assigning a public IP address.



```
DBSubnetGroup:
  Type: AWS::RDS::DBSubnetGroup
  Properties:
    DBSubnetGroupDescription: "Private subnets for RDS across 2 AZs"
    SubnetIds:
      - !Ref PrivateSubnetA
      - !Ref PrivateSubnetB
    Tags:
      - Key: Name
        Value: RDSPrivateSubnetGroup
```

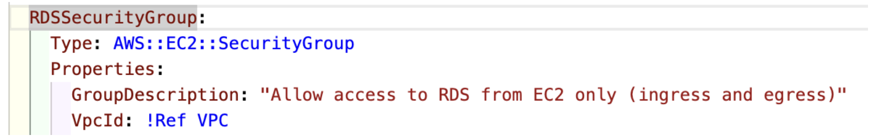


```
DBName: !Ref DBName
DBSubnetGroupName: !Ref DBSubnetGroup
VPCSecurityGroups:
  - !Ref RDSSecurityGroup
```



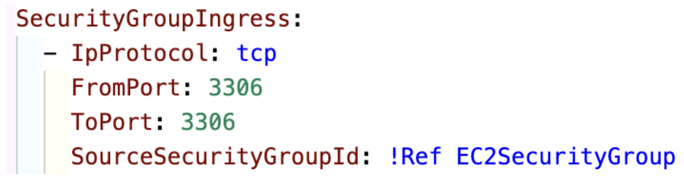
```
      - !Ref RDSSecurityGroup
    StorageEncrypted: true
    KmsKeyId: !Ref KMSKey
    BackupRetentionPeriod: 7
    PubliclyAccessible: false
```

- Create a security group for RDS in CloudFormation



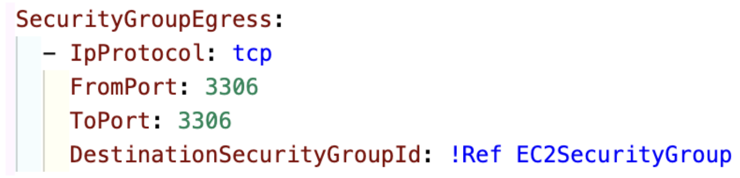
```
RDSSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: "Allow access to RDS from EC2 only (ingress and egress)"
    VpcId: !Ref VPC
```

- Configure an inbound rule to the RDS security group to allow TCP port 3306 traffic, with the source set to the EC2 instance's security group.



```
SecurityGroupIngress:
  - IpProtocol: tcp
    FromPort: 3306
    ToPort: 3306
    SourceSecurityGroupId: !Ref EC2SecurityGroup
```

- Configure an egress rule to the RDS security group to allow TCP port 3306 traffic, with the destination set to the EC2 instance's security group.



```
SecurityGroupEgress:
  - IpProtocol: tcp
    FromPort: 3306
    ToPort: 3306
    DestinationSecurityGroupId: !Ref EC2SecurityGroup
```

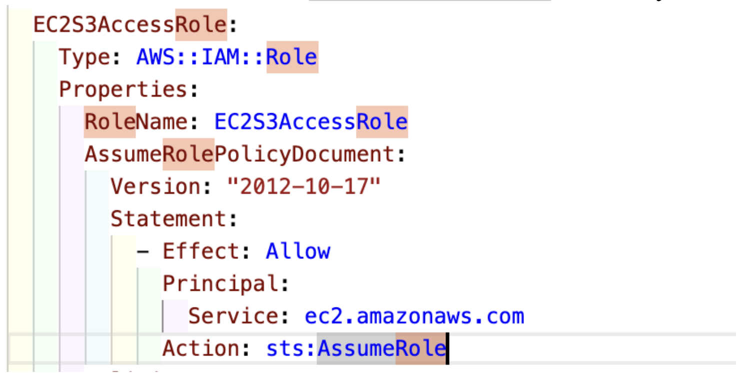
Improvement 3: Configure Bucket Policy(only allow ec2 instance to assume IAM role to access)

Purpose: Implement strict access controls on S3 buckets, allowing access only from EC2 instances associated with the corresponding IAM roles, and denying access to all other entities.

Tools/AWS Services: S3, IAM role, S3 Bucket Policy, CloudFormation

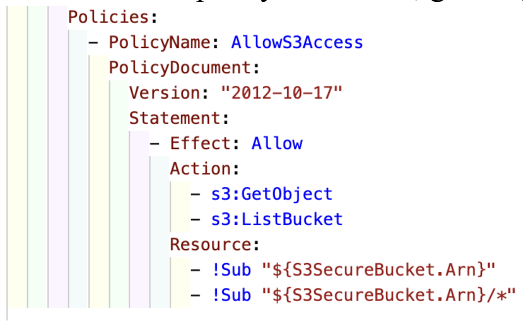
Technical Steps:

- Create an IAM role named EC2S3AccessRole that only allows EC2 instances to access



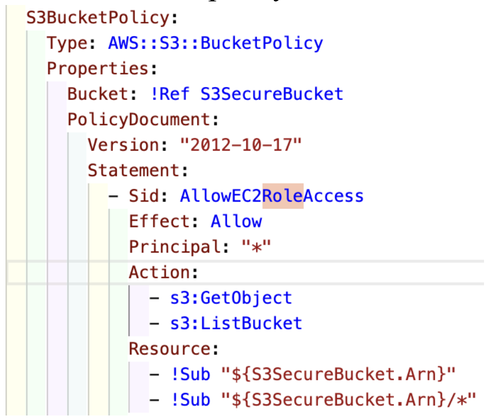
```
EC2S3AccessRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: EC2S3AccessRole
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
            Service: ec2.amazonaws.com
          Action: sts:AssumeRole
```

- Attach an inline policy to the role, granting GetObject and ListBucket permissions



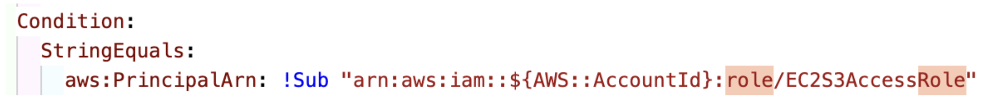
```
Policies:
  - PolicyName: AllowS3Access
    PolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Action:
            - s3:GetObject
            - s3:ListBucket
          Resource:
            - !Sub "${S3SecureBucket.Arn}"
            - !Sub "${S3SecureBucket.Arn}/*"
```

- Create a bucket policy that includes allow and deny access controls



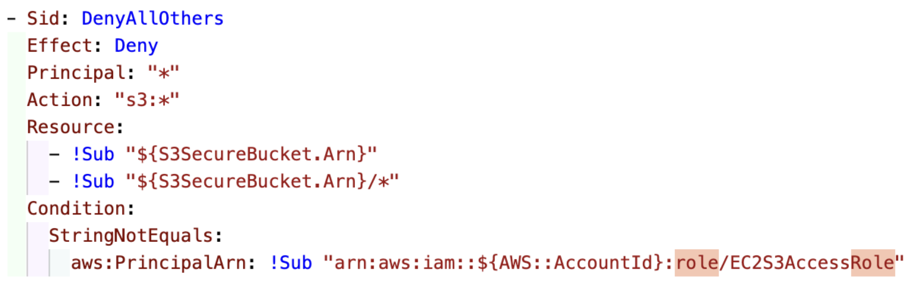
```
S3BucketPolicy:
  Type: AWS::S3::BucketPolicy
  Properties:
    Bucket: !Ref S3SecureBucket
    PolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Sid: AllowEC2RoleAccess
          Effect: Allow
          Principal: "*"
          Action:
            - s3:GetObject
            - s3:ListBucket
          Resource:
            - !Sub "${S3SecureBucket.Arn}"
            - !Sub "${S3SecureBucket.Arn}/*"
```

  Access is allowed only when the subject ARN matches EC2S3AccessRole.



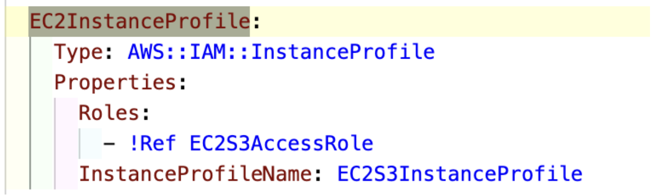
```
Condition:
  StringEquals:
    aws:PrincipalArn: !Sub "arn:aws:iam::${AWS::AccountId}:role/EC2S3AccessRole"
```

  If the principal is not this role (determined by StringNotEquals), all access is explicitly denied.



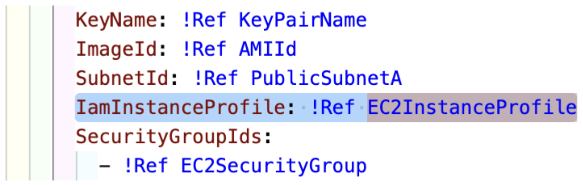
```yaml
- Sid: DenyAllOthers
  Effect: Deny
  Principal: "*"
  Action: "s3:*"
  Resource:
    - !Sub "${S3SecureBucket.Arn}"
    - !Sub "${S3SecureBucket.Arn}/*"
  Condition:
    StringNotEquals:
      aws:PrincipalArn: !Sub "arn:aws:iam::${AWS::AccountId}:role/EC2S3AccessRole"
```

- Bind the EC2 instance to the IAM role through the IamInstanceProfile in the EC2 instance configuration



```yaml
EC2InstanceProfile:
  Type: AWS::IAM::InstanceProfile
  Properties:
    Roles:
      - !Ref EC2S3AccessRole
    InstanceProfileName: EC2S3InstanceProfile
```

After creating the instance configuration file containing EC2S3AccessRole, we reference that configuration through the IamInstanceProfile field in the EC2 instance configuration, thereby binding the EC2 instance to the role and granting it secure access to S3.



```yaml
KeyName: !Ref KeyPairName
ImageId: !Ref AMIId
SubnetId: !Ref PublicSubnetA
IamInstanceProfile: !Ref EC2InstanceProfile
SecurityGroupIds:
  - !Ref EC2SecurityGroup
```

Validation and Testing

5.1 Validation

Network & Access Control Security

1. **Improvement 1: Harden security group for EC2 instance**

   We can check if the security group is hardened via checking the security group inbound and outbound rules. If appropriately hardened, the security group should not allow any other IP than the host IP, in order to prevent foreign network intrusions.

2. **Improvement 2: Harden Security group for RDS (only EC2 instance can access with SQL port)**

   We can check if the RDS security group is hardened by checking if we can access the SQL port (3306) from the EC2 instance's security group.

3. **Improvement 3: Configure Bucket Policy (only allow ec2 instance to assume IAM role to access)**

   We can check if the bucket policy only allows IAM roles with the specific role to access it via looking at the bucket policy section in the permissions tab of the s3 bucket.

<u>Data Protection & Encryption</u>

1. **Improvement 1: Instead of MariaDB on EC2 instance, use RDS**

   We can check if the instance is using RDS instead of MariaDB by checking the RDS dashboard for the appropriate database instance.

2. **Improvement 2: Encryption for S3, RDS, and EBS**

   We can check if the EBS volume is encrypted via navigating to the storage section, where it will show if the volume is encrypted via the Encrypted column. RDS encryption can be verified via checking the instance details of the relevant database instance. AWS S3 encryption can be confirmed via navigating to the properties of the s3 bucket, where it will state if server side encryption is enabled with AWS KMS.

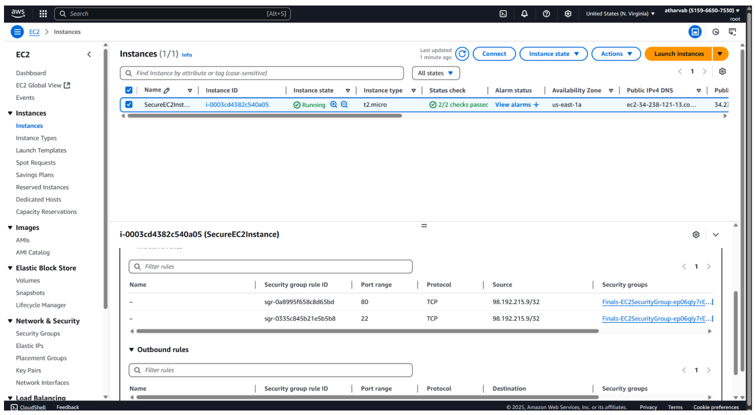
3. **Improvement 3: Use AWS KMS for Key management (Encryption key)**

   AWS KMS implementation can be checked via checking the key that is used to encrypt all the instances, and cross referencing it with the key that is available in the KMS customer managed keys section.

5.2 Evidence

<u>Network & Access Control Security</u>

1. **Improvement 1: Harden security group for EC2 instance**



Instances (1/1) Info

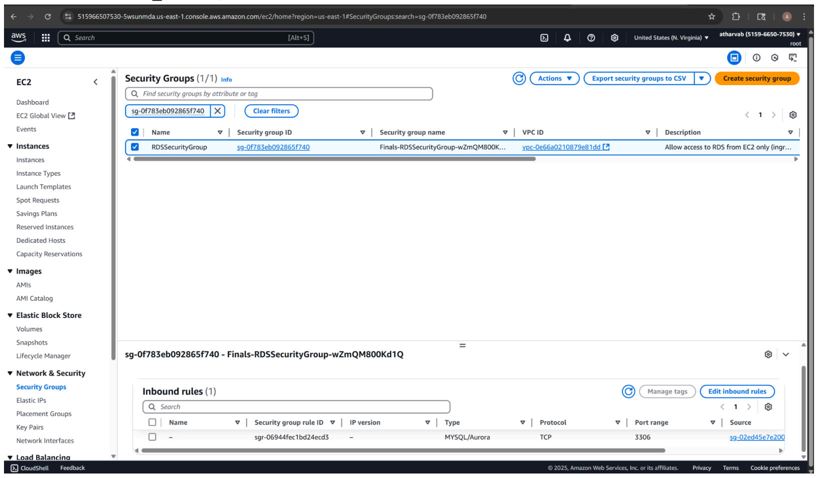
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Publi
SecureEC2Inst…	i-0003cd4382c540a05	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1a	ec2-34-238-121-13.co…	34.2:

i-0003cd4382c540a05 (SecureEC2Instance)

Name	Security group rule ID	Port range	Protocol	Source	Security groups
–	sgr-0a8995f658c8d65bd	80	TCP	98.192.215.9/32	Finals-EC2SecurityGroup-ep06qly7rE…
–	sgr-0335c845b21e5b5b8	22	TCP	98.192.215.9/32	Finals-EC2SecurityGroup-ep06qly7rE…

Name	Security group rule ID	Port range	Protocol	Destination	Security groups

2. **Improvement 2: Harden Security group for RDS (only EC2 instance can access with SQL port)**



Security Groups (1/1)

sg-0f783eb092865f740

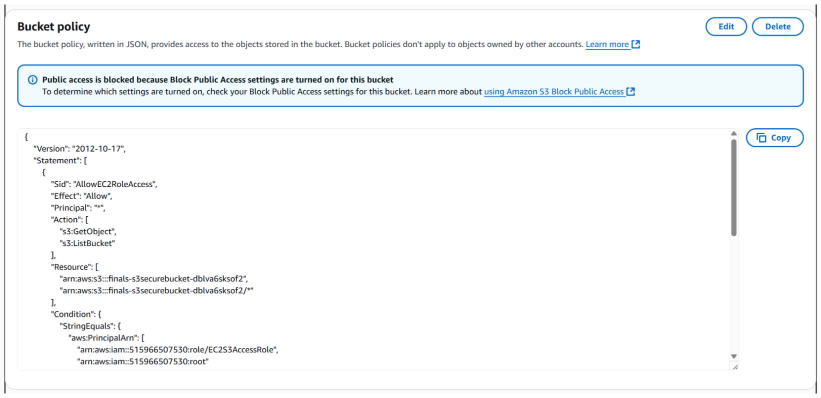
Name	Security group ID	Security group name	VPC ID	Description
RDSSecurityGroup	sg-0f783eb092865f740	Finals-RDSSecurityGroup-wZmQM800K...	vpc-0e66a0210879e81dd	Allow access to RDS from EC2 only (ingr...

sg-0f783eb092865f740 - Finals-RDSSecurityGroup-wZmQM800Kd1Q

Inbound rules (1)

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
–	sgr-06944fec1bd24ecd3	–	MYSQL/Aurora	TCP	3306	sg-02ed45e7e200

3. **Improvement 3: Configure Bucket Policy (only allow ec2 instance to assume IAM role to access)**



Bucket policy

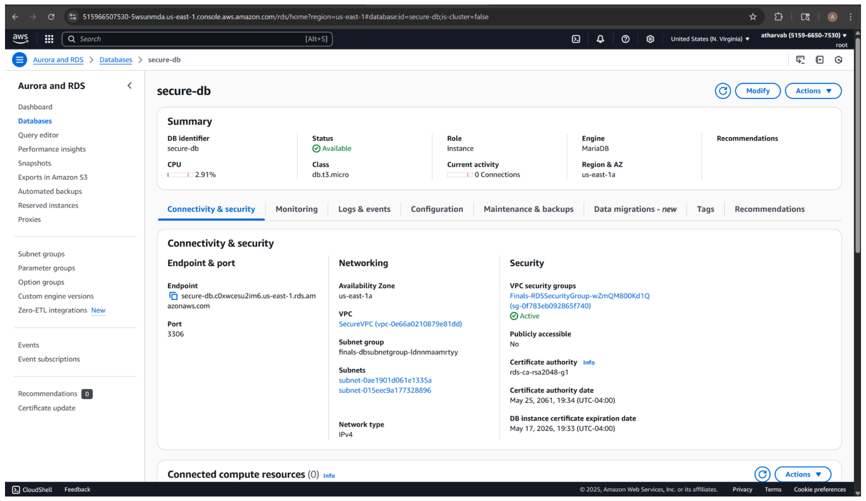
The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more

Public access is blocked because Block Public Access settings are turned on for this bucket
To determine which settings are turned on, check your Block Public Access settings for this bucket. Learn more about using Amazon S3 Block Public Access

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEC2RoleAccess",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::finals-s3securebucket-dblva6sksof2",
        "arn:aws:s3:::finals-s3securebucket-dblva6sksof2/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalArn": [
            "arn:aws:iam::515966507530:role/EC2S3AccessRole",
            "arn:aws:iam::515966507530:root"
```

<u>Data Protection & Encryption</u>

1. **Improvement 1: Instead of MariaDB on EC2 instance, use RDS**



secure-db

Summary

DB identifier	Status	Role	Engine	Recommendations
secure-db	Available	Instance	MariaDB	
CPU	**Class**	**Current activity**	**Region & AZ**	
2.91%	db.t3.micro	0 Connections	us-east-1a	

Connectivity & security

Endpoint & port

Endpoint
secure-db.c0xwcesu2im6.us-east-1.rds.amazonaws.com

Port
3306

Networking

Availability Zone
us-east-1a

VPC
SecureVPC (vpc-0e66a0210879e81dd)

Subnet group
finals-dbsubnetgroup-ldnnmaamrtyy

Subnets
subnet-0ae1901d061e1335a
subnet-015eec9a177328896

Network type
IPv4

Security

VPC security groups
Finals-RDSSecurityGroup-wZmQM800Kd1Q (sg-0f783eb092865f740)
Active

Publicly accessible
No

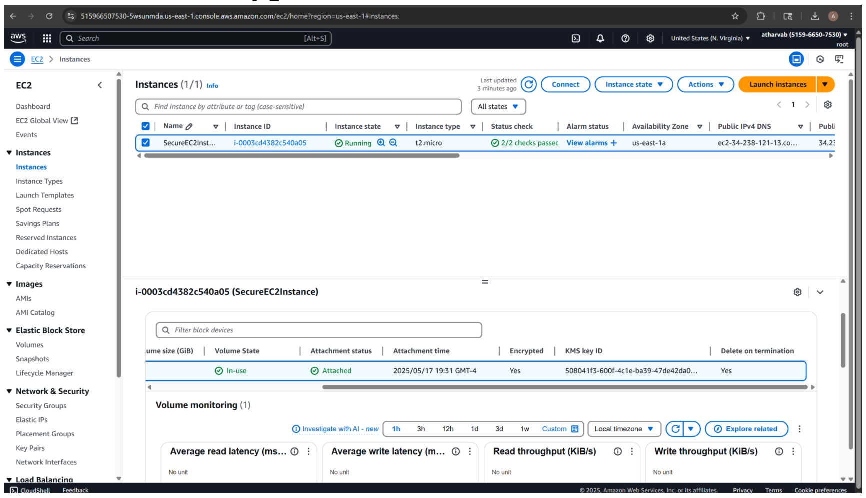
Certificate authority
rds-ca-rsa2048-g1

Certificate authority date
May 25, 2061, 19:34 (UTC-04:00)

DB instance certificate expiration date
May 17, 2026, 19:33 (UTC-04:00)

Connected compute resources (0)

2. Improvement 2: Encryption for S3, RDS, and EBS



Instances (1/1)

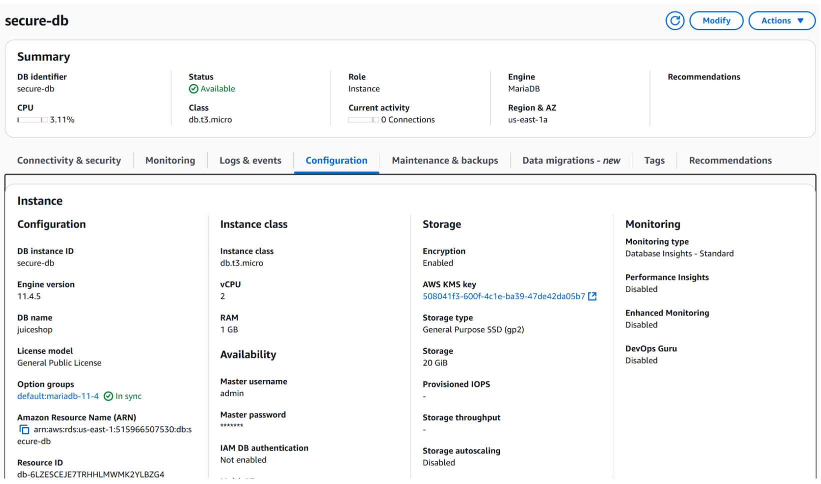
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Publi
SecureEC2Inst…	i-0003cd4382c540a05	Running	t2.micro	2/2 checks passec	View alarms	us-east-1a	ec2-34-238-121-13.co…	34.2

i-0003cd4382c540a05 (SecureEC2Instance)

ume size (GiB)	Volume State	Attachment status	Attachment time	Encrypted	KMS key ID	Delete on termination
	In-use	Attached	2025/05/17 19:31 GMT-4	Yes	508041f3-600f-4c1e-ba39-47de42da0…	Yes

Volume monitoring (1)

Average read latency (ms… No unit
Average write latency (m… No unit
Read throughput (KiB/s) No unit
Write throughput (KiB/s) No unit



secure-db

Summary

DB identifier	Status	Role	Engine	Recommendations
secure-db	Available	Instance	MariaDB	
CPU	**Class**	**Current activity**	**Region & AZ**	
3.11%	db.t3.micro	0 Connections	us-east-1a	

Instance

Configuration

DB instance ID
secure-db

Engine version
11.4.5

DB name
juiceshop

License model
General Public License

Option groups
default:mariadb-11-4 In sync

Amazon Resource Name (ARN)
arn:aws:rds:us-east-1:515966507530:db:secure-db

Resource ID
db-6LZESCEJE7TRHHLMWMK2YLBZG4

Instance class

Instance class
db.t3.micro

vCPU
2

RAM
1 GB

Availability

Master username
admin

Master password
****

IAM DB authentication
Not enabled

Storage

Encryption
Enabled

AWS KMS key
508041f3-600f-4c1e-ba39-47de42da05b7

Storage type
General Purpose SSD (gp2)

Storage
20 GiB

Provisioned IOPS
-

Storage throughput
-

Storage autoscaling
Disabled

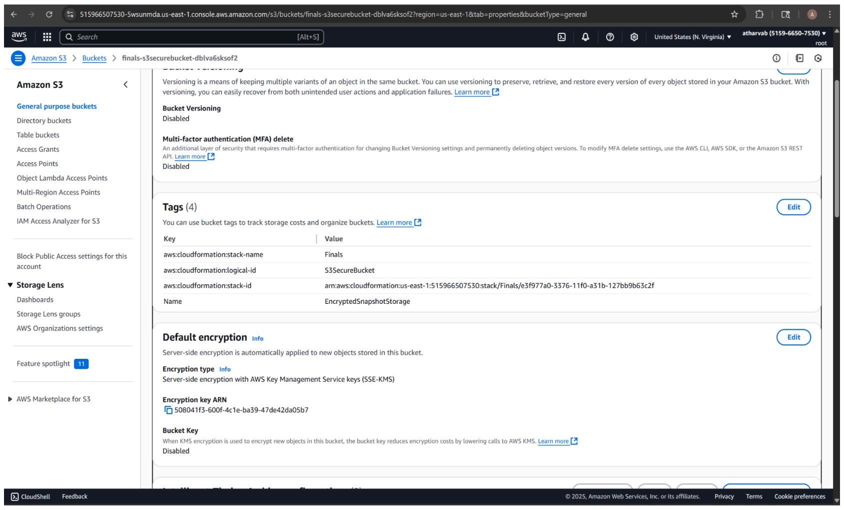
Monitoring

Monitoring type
Database Insights - Standard

Performance Insights
Disabled

Enhanced Monitoring
Disabled

DevOps Guru
Disabled



Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. Learn more

Bucket Versioning
Disabled

Multi-factor authentication (MFA) delete
An additional layer of security that requires multi-factor authentication for changing Bucket Versioning settings and permanently deleting object versions. To modify MFA delete settings, use the AWS CLI, AWS SDK, or the Amazon S3 REST API. Learn more
Disabled

Tags (4)
You can use bucket tags to track storage costs and organize buckets. Learn more

Key	Value
aws:cloudformation:stack-name	Finals
aws:cloudformation:logical-id	S3SecureBucket
aws:cloudformation:stack-id	arn:aws:cloudformation:us-east-1:515966507530:stack/Finals/e3f977a0-3376-11f0-a31b-127bb9b63c2f
Name	EncryptedSnapshotStorage

Default encryption
Server-side encryption is automatically applied to new objects stored in this bucket.

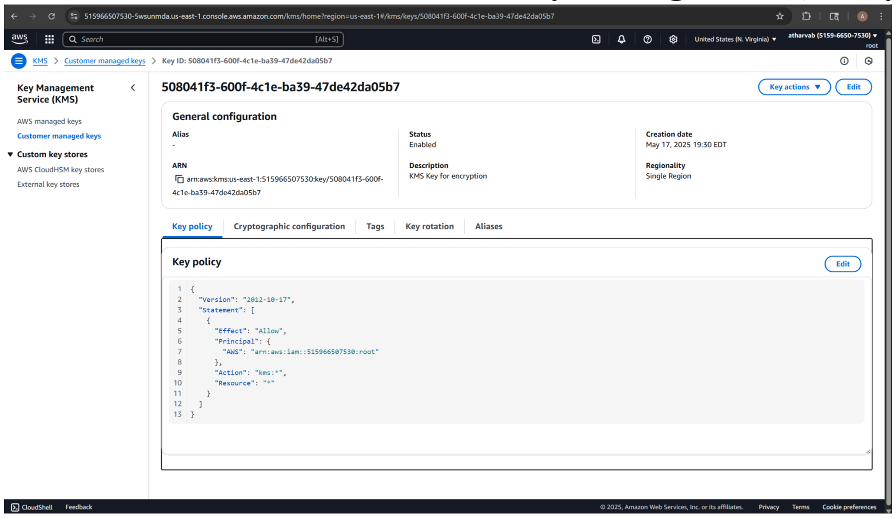
Encryption type
Server-side encryption with AWS Key Management Service keys (SSE-KMS)

Encryption key ARN
508041f3-600f-4c1e-ba39-47de42da05b7

Bucket Key
When KMS encryption is used to encrypt new objects in this bucket, the bucket key reduces encryption costs by lowering calls to AWS KMS. Learn more
Disabled

515966507530-5wsunmda.us-east-1.console.aws.amazon.com/s3/buckets/finals-s3securebucket-dblva6sksof2?region=us-east-1&tab=properties&bucketType=general
Search [Alt+S]
United States (N. Virginia)
atharvab (5159-6650-7530)
root
Amazon S3 > Buckets > finals-s3securebucket-dblva6sksof2
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 11
AWS Marketplace for S3
Edit
Info
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

3. Improvement 3: Use AWS KMS for Key management (Encryption key)



508041f3-600f-4c1e-ba39-47de42da05b7

General configuration

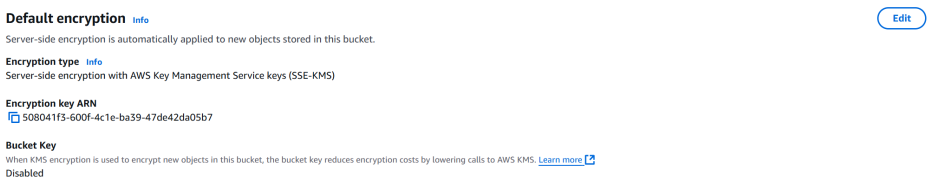
Alias -	**Status** Enabled	**Creation date** May 17, 2025 19:30 EDT
ARN arn:aws:kms:us-east-1:515966507530:key/508041f3-600f-4c1e-ba39-47de42da05b7	**Description** KMS Key for encryption	**Regionality** Single Region

Key policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::515966507530:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    }
  ]
}
```

515966507530-5wsunmda.us-east-1.console.aws.amazon.com/kms/home?region=us-east-1#/kms/keys/508041f3-600f-4c1e-ba39-47de42da05b7
Search [Alt+S]
United States (N. Virginia)
atharvab (5159-6650-7530)
root
KMS > Customer managed keys > Key ID: 508041f3-600f-4c1e-ba39-47de42da05b7
Key Management Service (KMS)
AWS managed keys
Customer managed keys
Custom key stores
AWS CloudHSM key stores
External key stores
Key actions
Edit
Key policy
Cryptographic configuration
Tags
Key rotation
Aliases
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

S3



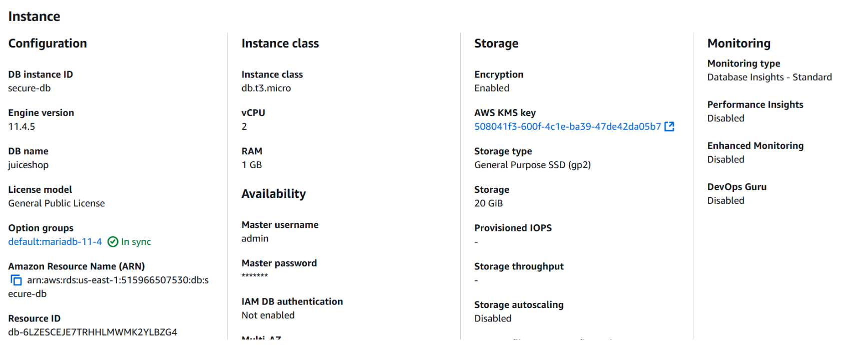
Default encryption
Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type
Server-side encryption with AWS Key Management Service keys (SSE-KMS)

Encryption key ARN
508041f3-600f-4c1e-ba39-47de42da05b7

Bucket Key
When KMS encryption is used to encrypt new objects in this bucket, the bucket key reduces encryption costs by lowering calls to AWS KMS. Learn more
Disabled

RDS



Instance

Configuration

DB instance ID
secure-db

Engine version
11.4.5

DB name
juiceshop

License model
General Public License

Option groups
default:mariadb-11-4 In sync

Amazon Resource Name (ARN)
arn:aws:rds:us-east-1:515966507530:db:secure-db

Resource ID
db-6LZESCEJE7TRHHLMWMK2YLBZG4

Instance class

Instance class
db.t3.micro

vCPU
2

RAM
1 GB

Availability

Master username
admin

Master password
****

IAM DB authentication
Not enabled

Storage

Encryption
Enabled

AWS KMS key
508041f3-600f-4c1e-ba39-47de42da05b7

Storage type
General Purpose SSD (gp2)

Storage
20 GiB

Provisioned IOPS
-

Storage throughput
-

Storage autoscaling
Disabled

Monitoring

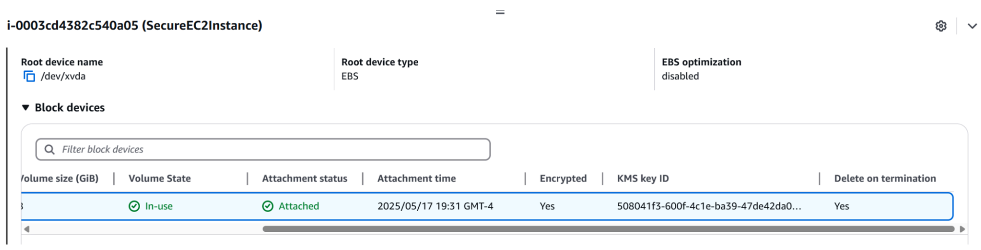
Monitoring type
Database Insights - Standard

Performance Insights
Disabled

Enhanced Monitoring
Disabled

DevOps Guru
Disabled

EBS



i-0003cd4382c540a05 (SecureEC2Instance)

Root device name
/dev/xvda

Root device type
EBS

EBS optimization
disabled

Block devices

Volume size (GiB)	Volume State	Attachment status	Attachment time	Encrypted	KMS key ID	Delete on termination
	In-use	Attached	2025/05/17 19:31 GMT-4	Yes	508041f3-600f-4c1e-ba39-47de42da0...	Yes

Impact Assessment

This cloud security initiative resulted in a significant uplift in the security posture of our AWS-based e-commerce platform. Targeted remediations addressed critical risks in Data Protection & Encryption and Network & Access Control Security, transforming the architecture from an open, high-risk design into a segmented, encrypted, and auditable cloud environment aligned with AWS best practices and NIST Cybersecurity Framework principles.

6.1 Data Protection & Encryption

Before Remediation:

- EC2 volumes had no encryption defined (Encrypted: false), exposing data to unauthorized snapshot recovery.
- Database credentials were embedded in EC2 UserData scripts and stored unencrypted on disk.
- The S3 bucket used for backup storage had no explicit BucketEncryption policy.

After Remediation:

- EBS volumes were configured with:

```
Encrypted: true
KmsKeyId:!Ref KMSKey
```

- RDS MariaDB is now encrypted at rest (StorageEncrypted: true) and made non-public (PubliclyAccessible: false).

```
StorageEncrypted: true
KmsKeyId: !Ref KMSKey
PubliclyAccessible: false
```

- Credentials are now stored in AWS Secrets Manager, auto-generated and injected via:

```
{{resolve:secretsmanager:${SecretForDBPassword}::password}}
```

- S3 bucket uses AWS KMS encryption:

```
SSEAlgorithm: aws:kms
KMSMasterKeyID: !Ref KMSKey
```

Impact:

- All data at rest (EBS, RDS, S3) is now encrypted using customer-managed KMS keys.
- Secrets are securely managed and no longer exposed in instance metadata or file system.
- Key management is centralized and auditable.

Residual Risks & Recommendations:

- Ensure regular key rotation policies are enforced in KMS.
- Monitor Secrets Manager access via CloudTrail.
- Implement alerts for unusual access patterns or failed decryption attempts.

6.2 Network & Access Control Security

Before Remediation:

- Flat VPC with all components in public subnets.
- Security groups allowed unrestricted inbound SSH (port 22) and HTTP (port 80).
- RDS was publicly accessible with no strict traffic controls.

After Remediation:

Introduced segmented VPC with:

- **PublicSubnetA for EC2** `(10.0.1.0/24)`
- **PrivateSubnetA/B for RDS** `(10.0.2.0/24, 10.0.3.0/24)`

Hardened Security Groups:

- **RDS only allows traffic from EC2 SG**
- **RDS make** `PubliclyAccessible: false`

Impact:

- Reduced attack surface by isolating application and data layers.
- Least privilege enforced at both subnet and security group levels.
- Improved infrastructure alignment with zero-trust network design principles.

Residual Risks & Recommendations:

- SSH remains open for practical evaluation access, but the design supports future transition to a controlled bastion host model.
- Add AWS WAF and ALB in future to control HTTP/S traffic and add DDoS protection.
- Enable VPC Flow Logs and GuardDuty for deeper visibility into network events.

Trade-offs & Operational Notes

- KMS and Secrets Manager introduce minor cost overhead but drastically improve security.
- Encryption and segmentation may slightly increase complexity, but greatly reduce lateral movement risks.
- All changes remain within the scope of AWS Free Tier or low-tier resource usage for demonstration.

Lessons learned and Group Reflection

7.1 Key Lessons Learned

IAM Integration Is Foundational

Many critical services, including Secrets Manager and KMS, depended on IAM role trust relationships and scoped permissions. Missteps in IAM policies especially those tied to secret access or encryption key usage proved to be a common bottleneck during deployment testing.

Carefully configure IAM roles and S3 bucket policies

We created a separate IAM role for EC2 instances to access S3 and explicitly denied all access from roles other than this role in the bucket policy. Although this improved security, it also introduced operational risks—when the role was deleted, we were unable to access or delete the bucket and ultimately had to recreate and assume the role with the same name to unlock the resources. This process reminded us that security policies must balance permission control and operational feasibility.

Hands-On Documentation Saves Time

Capturing YAML configurations, deployment logs, and architecture decisions made the debugging process more efficient and ensured continuity during handoff or review. Reusability and reproducibility improved substantially due to deliberate documentation practices.

7.2 Suggestions for Future Improvements

Master CloudFormation (Infrastructure-as-Code)

We have effectively used AWS CloudFormation to automate infrastructure deployment in this improvement project. However, in future versions, we can further improve deployment efficiency and consistency by modularizing resources and introducing CI/CD processes. That will not only enhance infrastructure repeatability, but also make the update process safer and more controllable with version control.

Enable CloudWatch and CloudTrail

Although we have implemented multiple encryption and access control mechanisms in this implementation, these measures alone cannot guarantee the absolute security of the cloud environment. Therefore, enabling CloudWatch and CloudTrail is essential. The former provides continuous monitoring of the operational status of all cloud resources, while the latter records all API calls. When incidents arise, these tools will quickly identify issues and provide support.

Introduce Auto Scaling and High Availability Features

This security enhancement does not yet address fault tolerance. However, commercial websites facing customers in real-world scenarios require high availability and disaster recovery capabilities. In the future, elastic scaling and automatic fault recovery can be achieved through the implementation of mechanisms such as Auto Scaling Groups, EC2 health checks, and multi-availability zone deployment at the web layer, thereby improving the overall availability of the service.

Encrypt data in transit

Encrypting data during transmission was included in our early planning as part of data encryption, but in this iteration, we have not yet fully implemented TLS-encrypted communication between EC2 and RDS. In the future, we should prioritize configuring and verifying TLS to ensure that data cannot be intercepted or tampered with during transmission.