STRENGTHENING AWS DATA SECURITY: A COMPLETE FRAMEWORK FOR EBS DATA PROTECTION AND ENCRYPTION

FEBRUARY 25, 2025

SHAISTHA KHANUM

Table of Contents

Introduction:

In the era of cloud computing, data security has become a cornerstone of resilient and compliant infrastructure. Amazon Elastic Block Store (EBS), a vital component of Amazon Web Services (AWS), offers high-performance block storage for EC2 instances, supporting a wide range of workloads—from critical databases to large-scale analytics. However, the sensitive nature of data stored on EBS volumes demands robust protection measures to safeguard against unauthorized access, data breaches, and compliance violations.

This report presents a comprehensive framework for strengthening AWS data security by focusing on EBS data protection and encryption. It explores native AWS encryption mechanisms, highlights the role of AWS Key Management Service (KMS) in securing data at rest, and outlines best practices for managing encryption keys. The report also provides a step-by-step guide for encrypting existing unencrypted EBS volumes, leveraging automation with Terraform and Bash scripts, and implementing security compliance checks using tools like Checkov.

Furthermore, it delves into the performance impact of EBS encryption, compliance considerations, and strategies for effective data recovery and incident response. By adopting this end-to-end approach, organizations can ensure that their AWS environments not only meet regulatory standards but also uphold the highest levels of data confidentiality, integrity, and availability.

Amazon Elastic Block Storage

Amazon Elastic Block Storage (EBS) is a block-level storage service designed for use with Amazon EC2 instances. Acting like a virtual SSD, EBS provides scalable and high-performance storage for various applications, including databases, big data analytics engines, and web applications. Given the sensitive nature of the data often stored on EBS volumes, it's critical to implement robust data protection strategies, including encryption.

Encrypting EBS volumes

By default, EBS volumes are not encrypted when created unless configured otherwise in the EC2 settings. Once an EBS volume is created, its encryption status cannot be altered. To change the encryption state, data must be migrated to a new volume with the desired encryption settings.

Key Points:

- Snapshots created from encrypted volumes are automatically encrypted.
- EBS encryption is transparent to end-users and applications.
- While EBS encryption protects against unauthorized access to physical disks, it doesn't safeguard data from threats within the EC2 instance.

Understanding AWS KMS and EBS Encryption

AWS Key Management Service (KMS) plays a central role in EBS encryption. It manages encryption keys and integrates with EBS for seamless encryption processes. Users can choose between AWS-managed keys or customer-managed keys (CMKs) for greater control.

- **AWS KMS Features:**
    - Centralized key management for EBS and other AWS services.
    - Fine-grained access control using AWS IAM policies.
    - Automatic key rotation for enhanced security.
    - Integration with AWS CloudTrail for auditing key usage and changes.
    - Support for cross-account access and multi-region key replication.

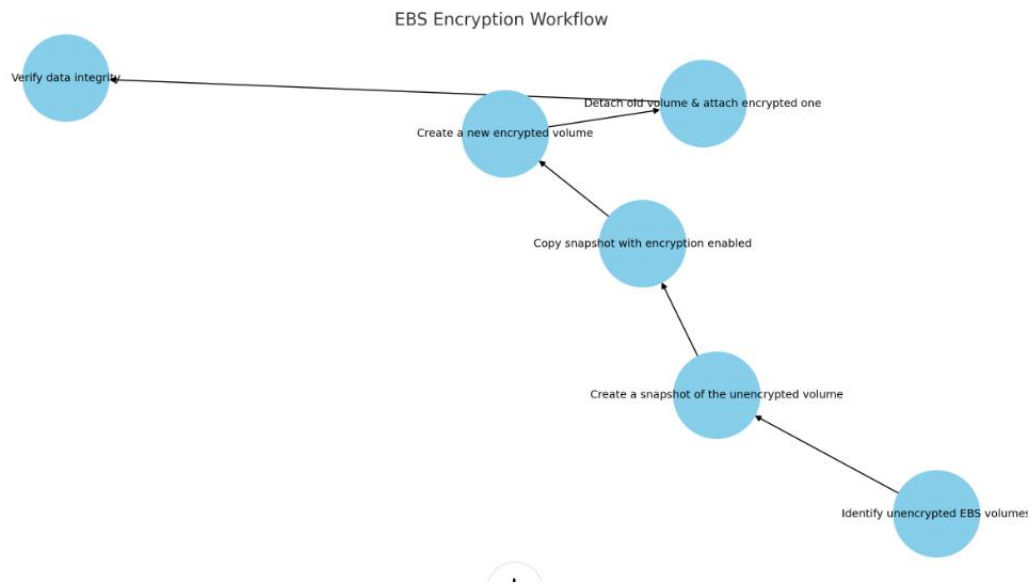
Encrypt Existing Unencrypted EBS Volumes and Snapshots

Learn how to manage unencrypted EBS volumes and snapshots attached to existing EC2 instances by following the correct process to encrypt them effectively.

Why EBS volumes/snapshots should be encrypted (and what that means)

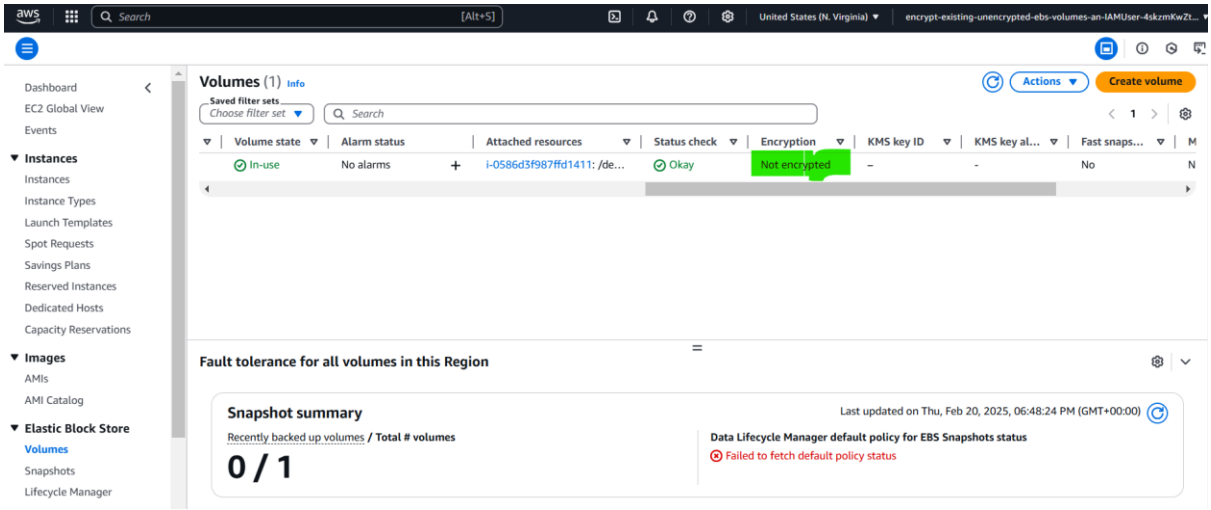
The TL;DR: This type of encryption protects against threats related to physical access to the actual disks. However, if a malicious actor gains access to your EC2 instance and successfully impersonates a legitimate user, they can still access the decrypted data—encryption at this level won't protect against that. Instead, encrypting volumes and snapshots primarily safeguards against rogue AWS employees or individuals with physical access to the underlying storage from viewing your data in plaintext.

Flow Chart



Identify unencrypted volumes:

1. Navigate to the EC2 Console.
2. In the navigation pane, select **Volumes** under **Elastic Block Store**.
3. Check the **Encryption** column for volumes marked as **Not Encrypted**.



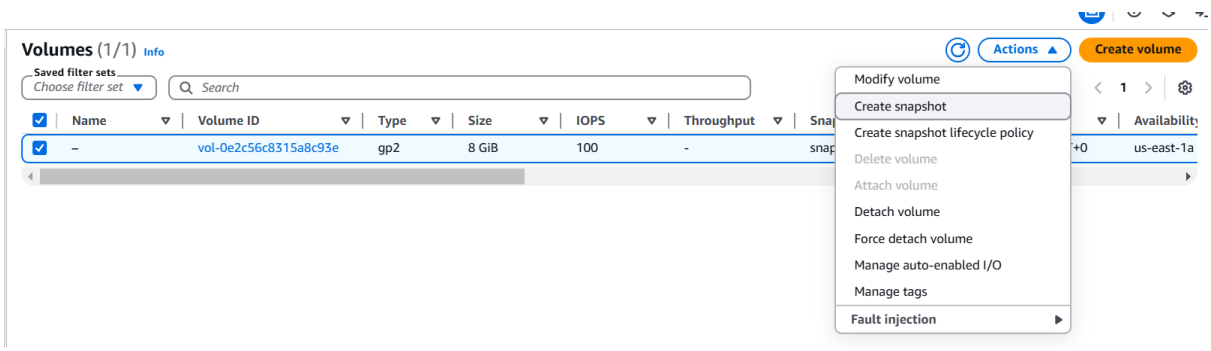
Create a snapshot of the unencrypted volume:

It's not possible to directly encrypt unencrypted volumes in AWS, so we first need to create a snapshot of the current volume so that we can then either create a new snapshot or volume from that snapshot and encrypt that new volume or snapshot.

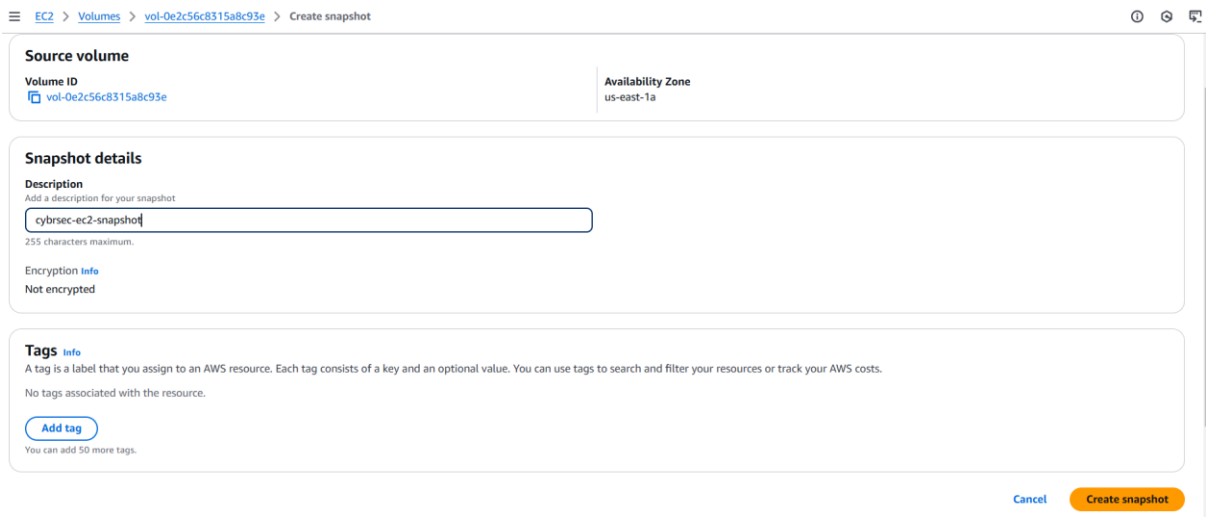
In this lab, we'll demonstrate creating an encrypted snapshot from an unencrypted snapshot, and then creating an encrypted volume from that new encrypted snapshot. But keep in mind you could technically skip that step and create an encrypted volume from an unencrypted snapshot, if you wanted to.

Let's get started:

1. Select the unencrypted volume.
2. Choose **Actions > Create Snapshot**.



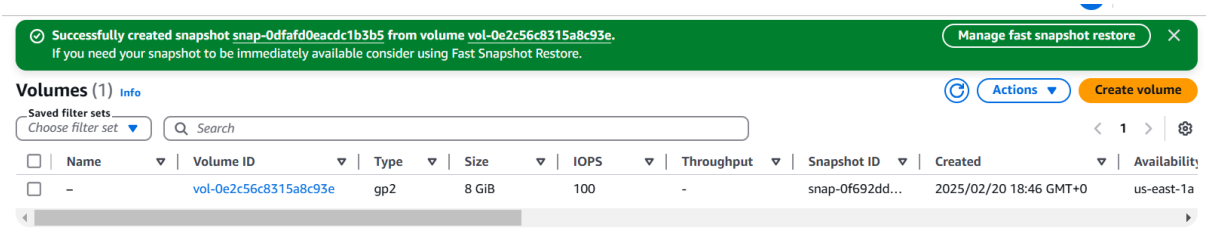
3. Enter a descriptive name and description for the snapshot.



4. Click Create Snapshot.

   Note: If you're not familiar, EBS snapshots are point-in-time copies of EBS volumes which can be used to create incremental backups of your data. These snapshots are stored in S3 behind the scenes (you can't see the buckets in your account) and can be used to restore data.

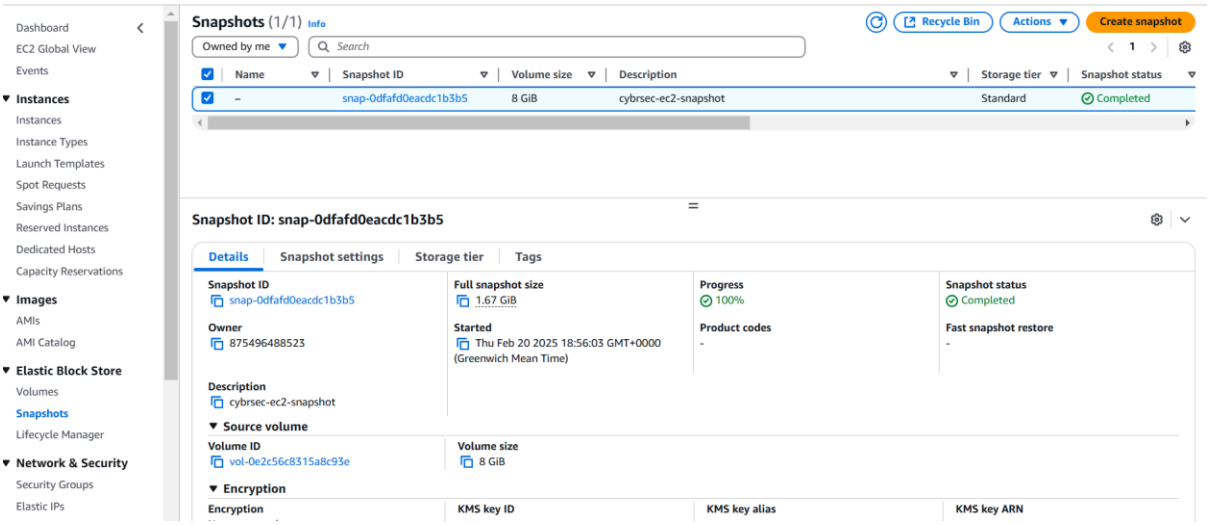
5. You'll see the green banner once you select Create



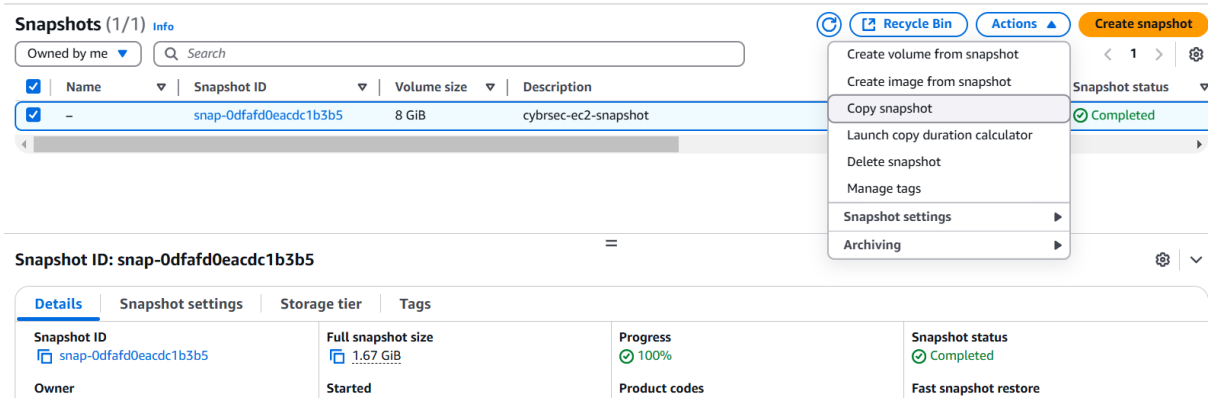
Take note of the snapshot ID that starts with snap-123xx. You may click on the snapshot ID as well. (Mine look like this: **snap-0dfafd0eacdc1b3b5** **and volume id: vol-0e2c56c8315a8c93e**.)

Copy the snapshot to an encrypted form:

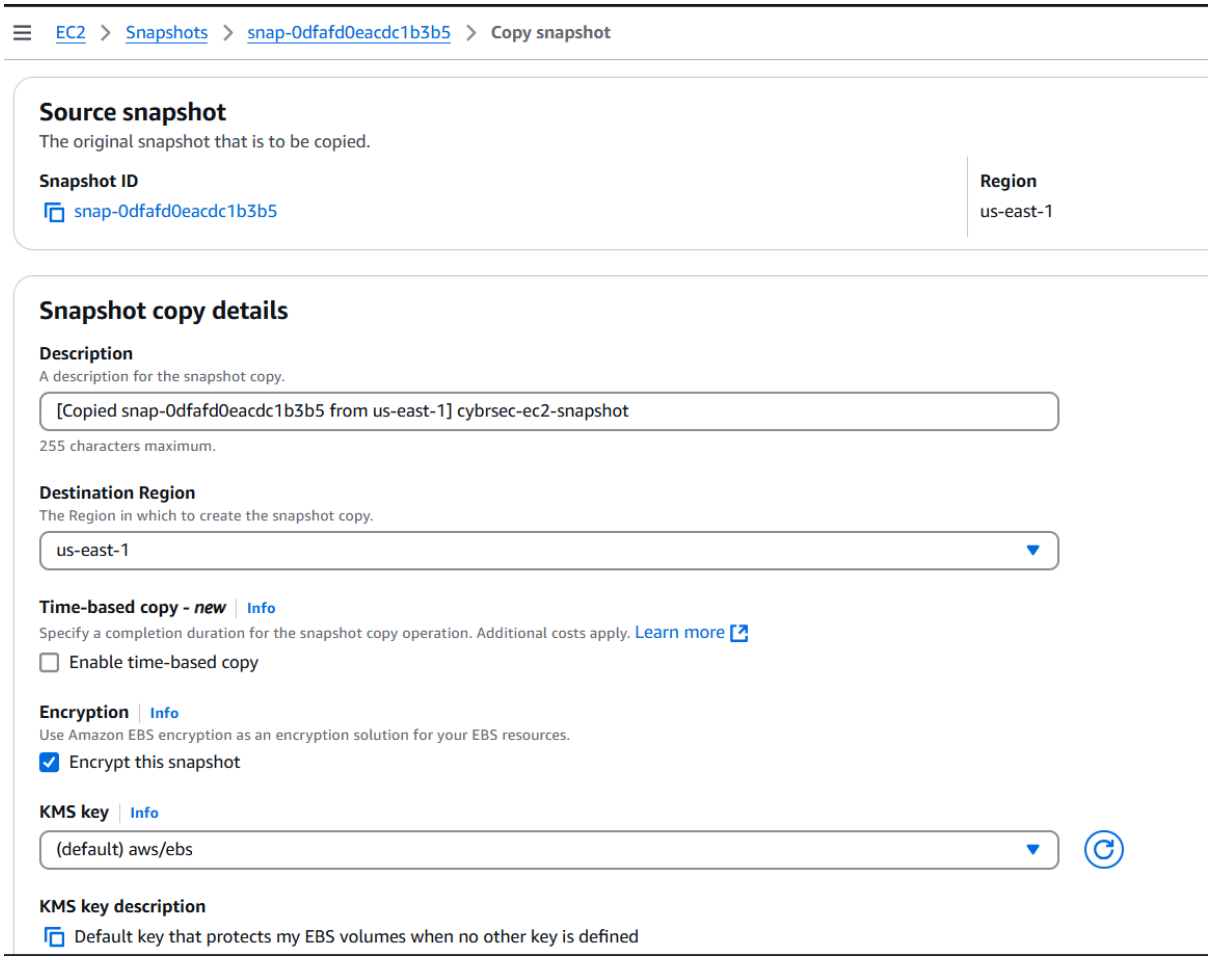
1. Go to **Snapshots** in the EC2 Console.
2. Select the snapshot and choose **Actions > Copy Snapshot**.



3. Set the **Destination Region** and check the **Encrypted** box.
4. Use the AWS-managed key (default) or a customer-managed key.



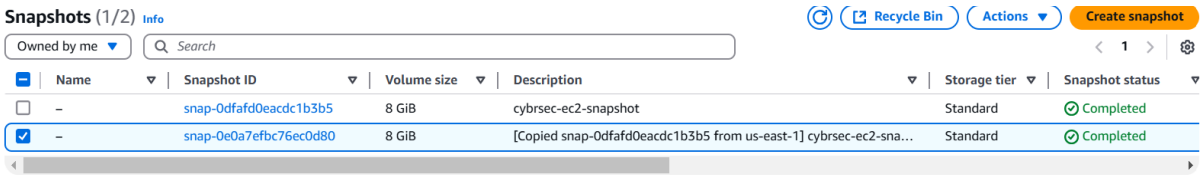
5. Click **Copy Snapshot**.



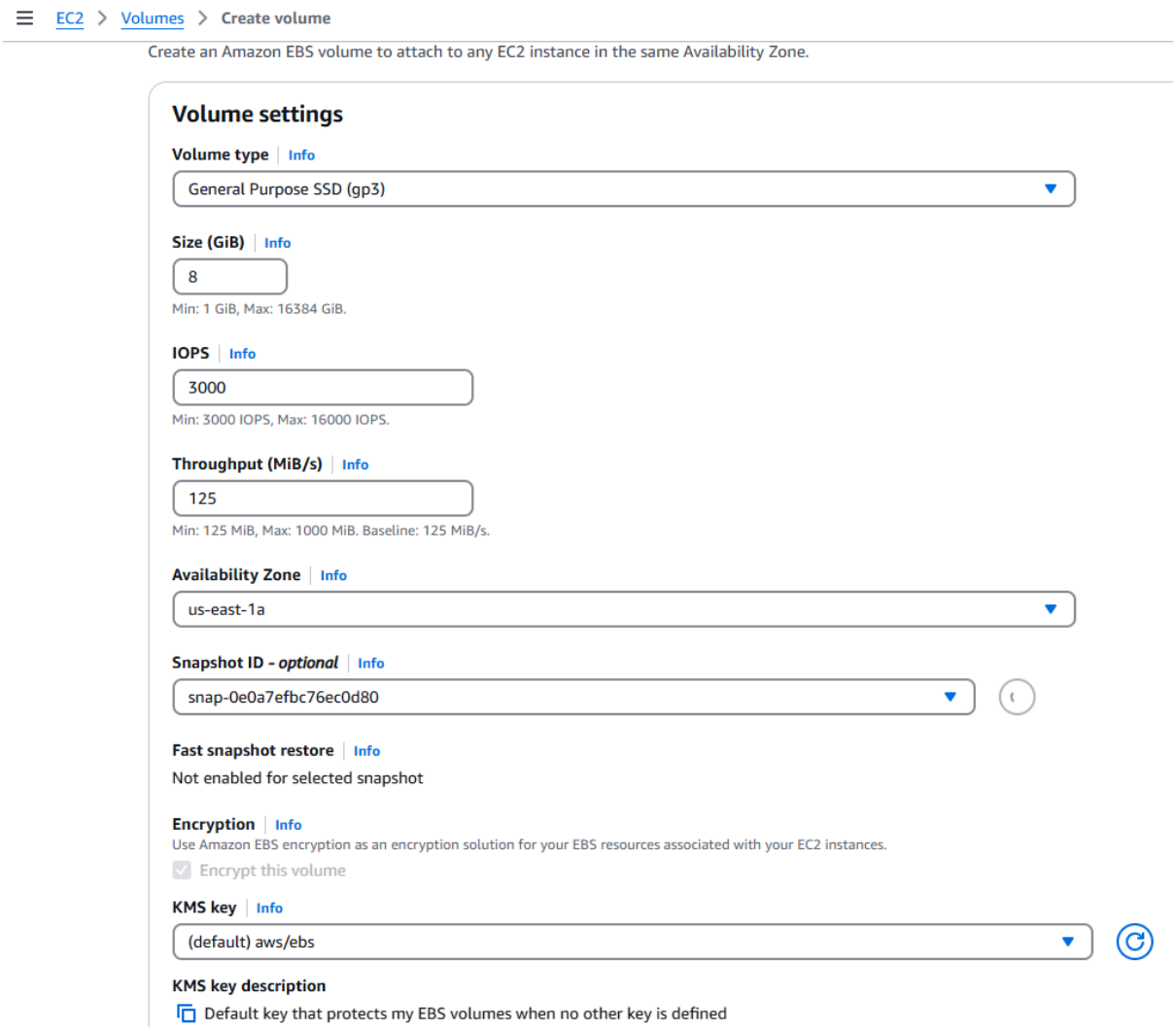
Note: If you were to choose to use a customer-managed key in your own environment, just know CMS keys incur additional costs for AWS KMS usage, while AWS-managed keys are included in the standard EBS pricing.

Create a new encrypted volume from the encrypted snapshot:

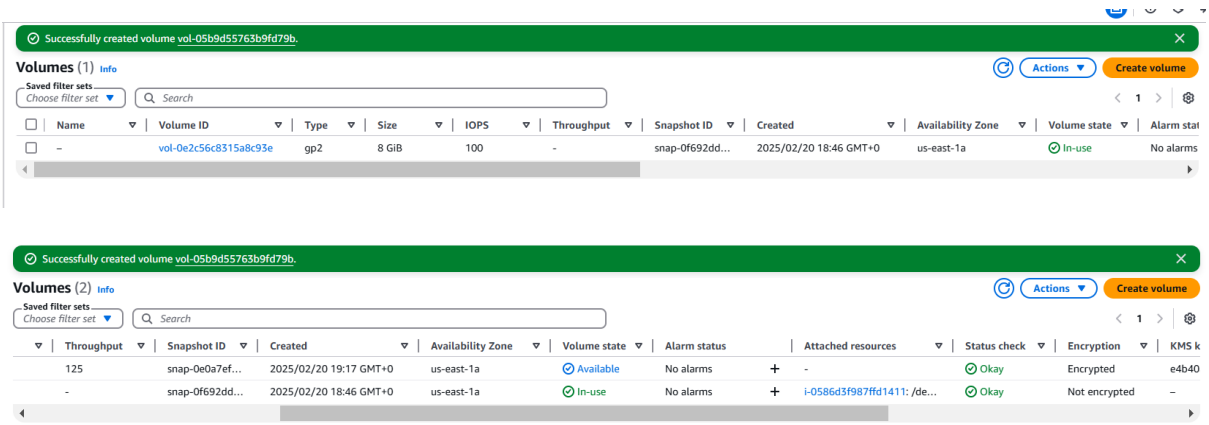
1. Wait for the encrypted snapshot to complete.
2. Before moving on, take note of the new snapshot ID that you just created (the one with Copied ... in the description) and copy/paste it in your notes.

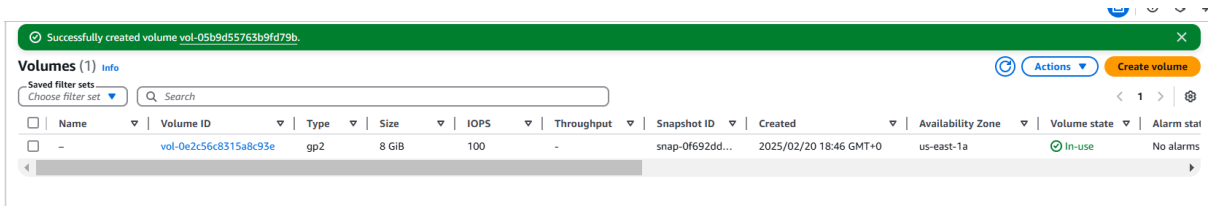


3. In **Volumes**, click **Create Volume**.
4. For the Size (GiB), enter 8 instead of 100



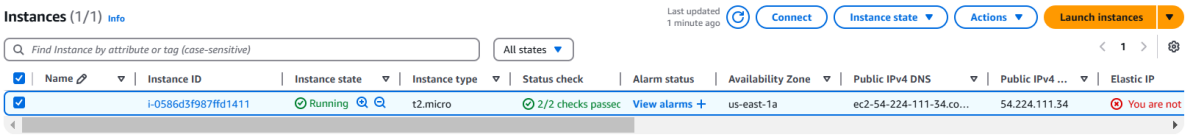
5. Select the encrypted snapshot, set the size and availability zone.
6. Ensure **Encrypt this volume** is selected.
7. Click **Create Volume**.

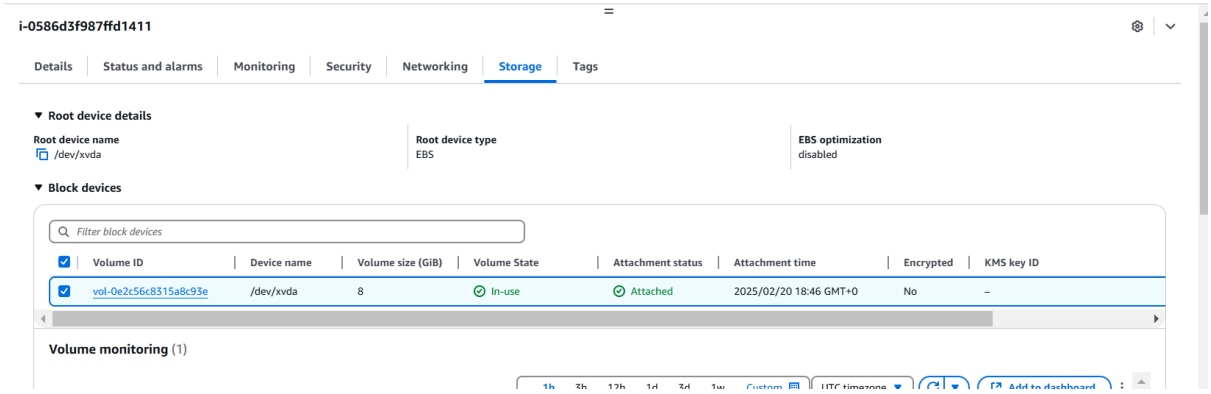




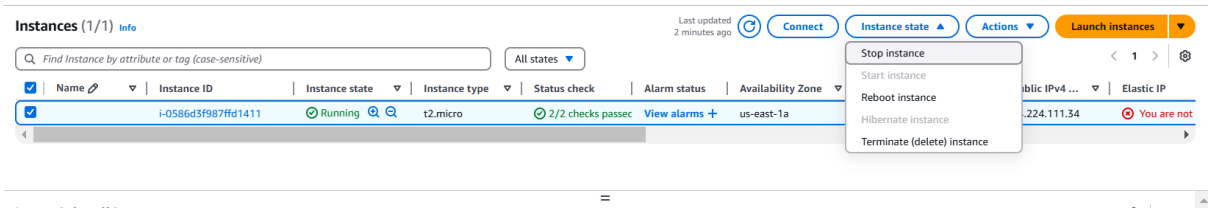
Detach the unencrypted volume from the EC2 instance:

1. Go to the EC2 console and select the instance that has the unencrypted volume attached
2. From the “Storage” section, find the unencrypted volume and note its device name (e.g., /dev/xvda)



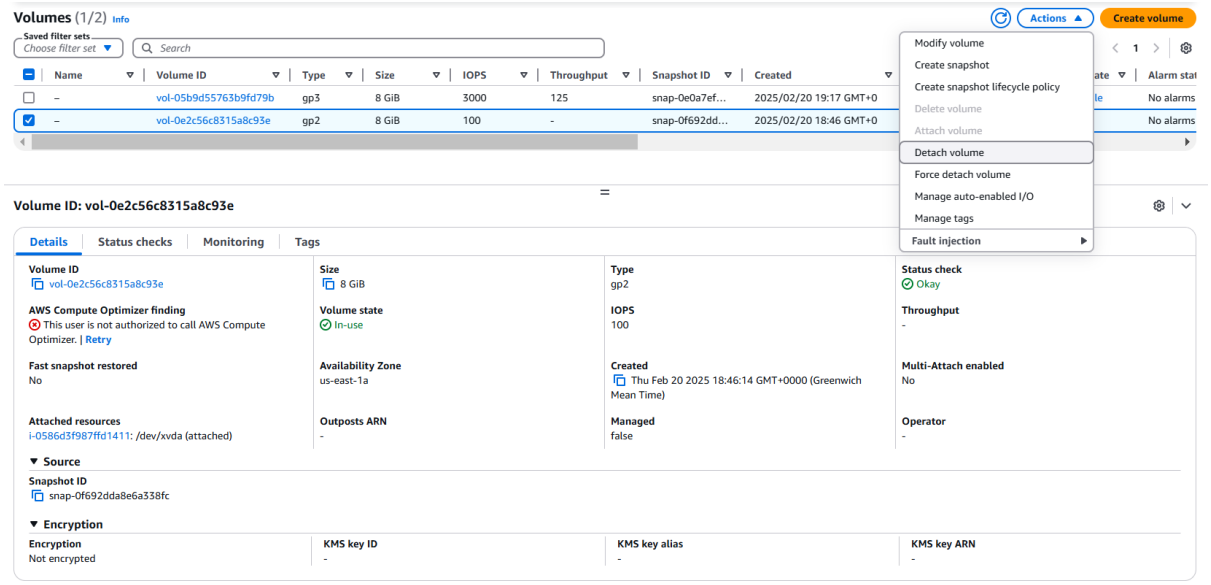


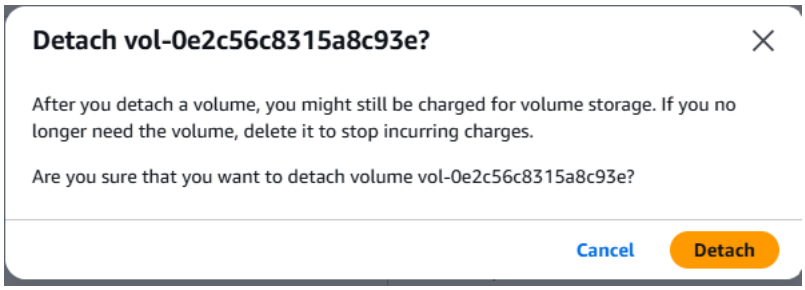
3. **Stop** the EC2 instance, choose Instance State > Stop Instance. Confirm.



4. Once the instance is stopped, navigate back to the Volumes dashboard
5. You may have to refresh this page to view both volumes if you only see one

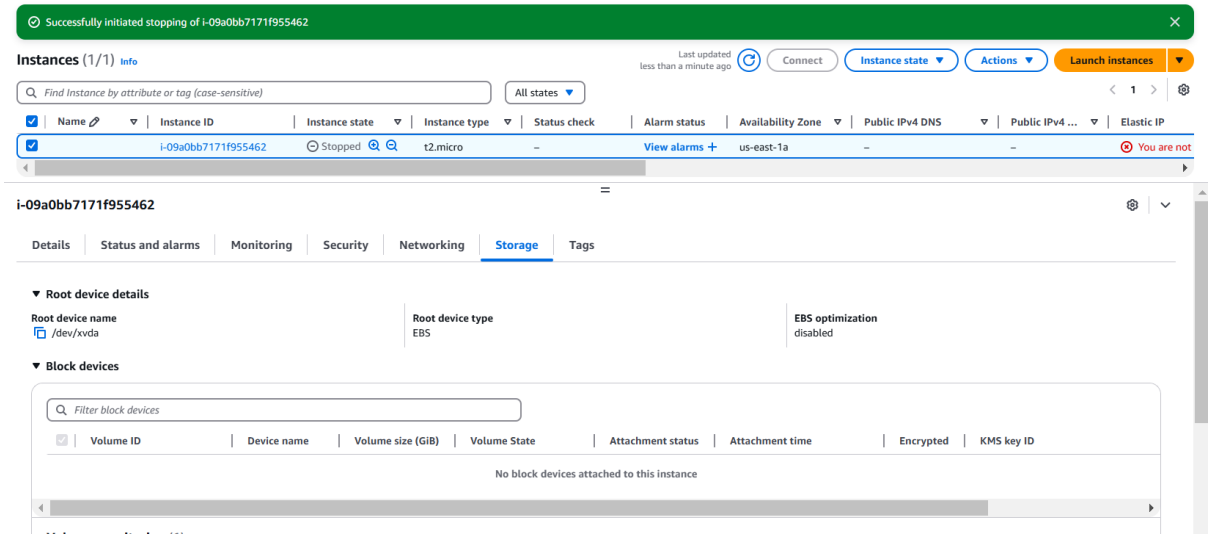
6. Detach the unencrypted volume via **Actions > Detach Volume**.
7. Confirm detachment.





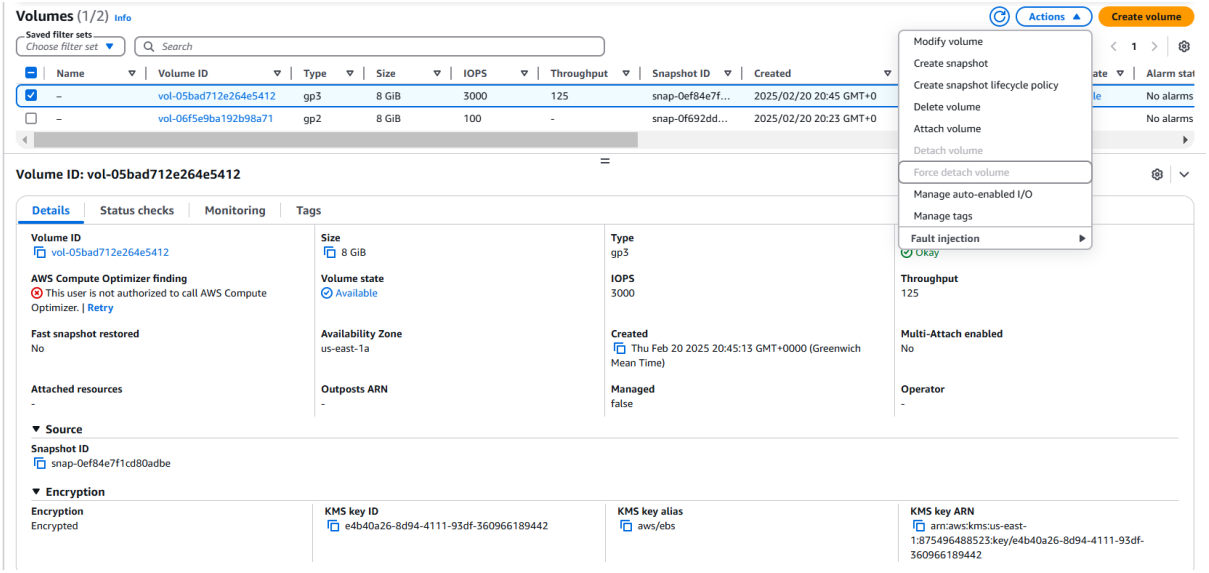
Attach the new encrypted volume to the EC2 instance:

1. Back in the EC2 console, select the stopped instance.
2. You'll now see that there are no Block devices attached.

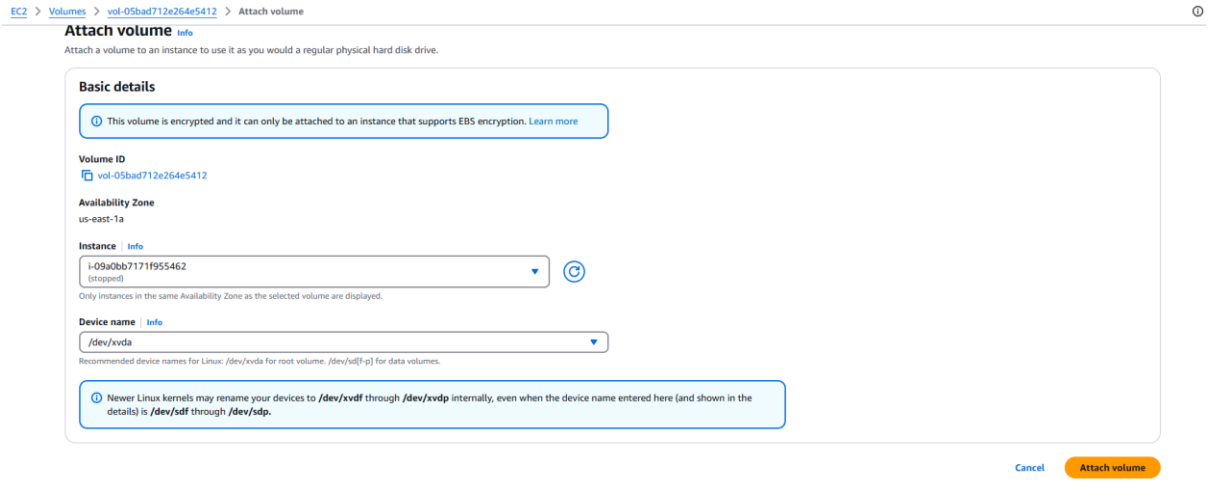


3. Go back to the Volumes dashboard
4. Select the new encrypted volume

5. Click on Actions > Attach Volume



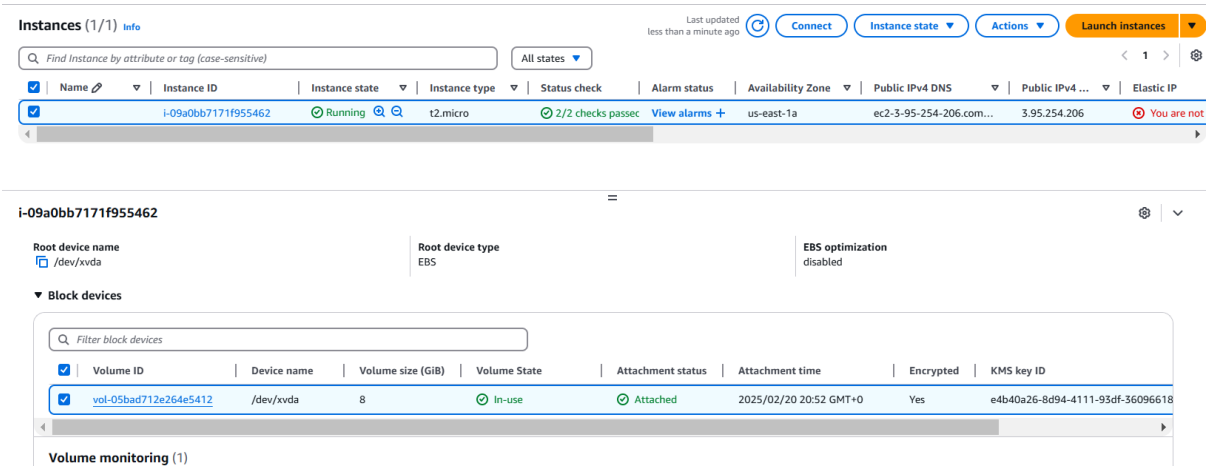
6. Select the instance from the dropdown
7. Choose the device name that was previously used by the unencrypted volume (it should be /dev/xvda).
8. Click Attach Volume.



Now we can go back to the EC2 instance, refresh our EC2 page, and see that our volume is attached.

Start the EC2 instance:

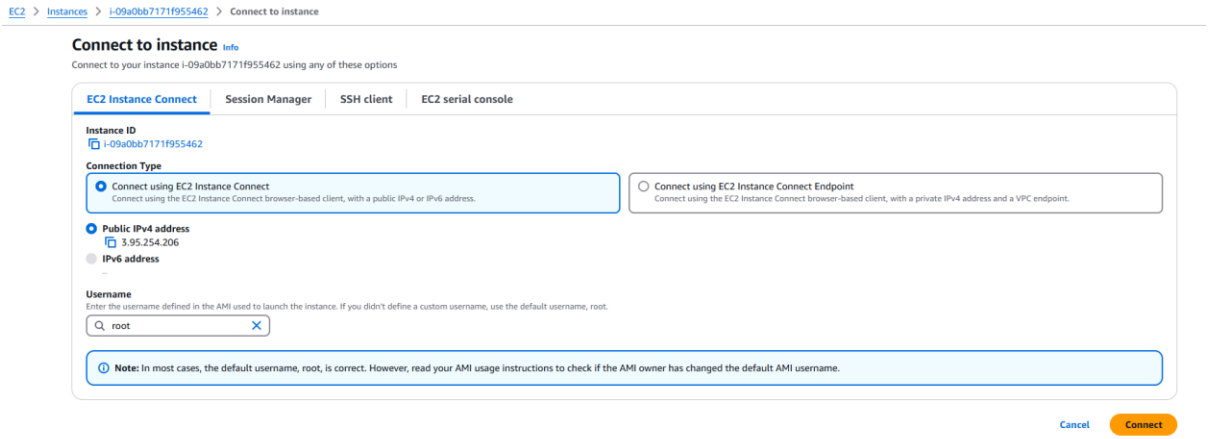
1. In the EC2 console, select the instance
2. Choose Instance State > Start Instance



Verify data integrity and encryption:

You'll need to wait a minute or so for the instance to boot up before you can connect to it, and you'll know it's ready when the Status check no longer says Initializing and instead shows a green 2/2 checks passed.

1. Connect to the EC2 instance using your preferred method (e.g., SSH or SSM).



2. On the EC2 Instance Connect tab, leave the defaults, ignore the warning and click on Connect.
3. Check if the data on the new encrypted volume is intact and accessible.
4. To start, you can use the df utility like this in the web-based terminal: # df -h


```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[root@ip-10-0-0-203 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  440K  190M   1% /run
/dev/xvda1      8.0G  1.6G  6.4G  21% /
tmpfs           475M     0  475M   0% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs            95M     0   95M   0% /run/user/0
[root@ip-10-0-0-203 ~]#
```

We'll see our /dev/xvda1 filesystem.

You can also use lsblk like this:

lsblk

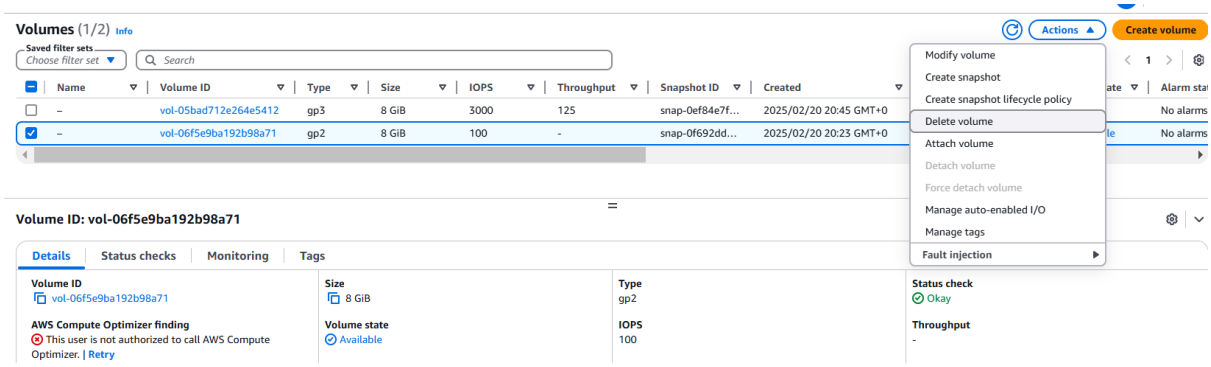
```
[root@ip-10-0-0-203 ~]# lsblk
NAME      MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
xvda      202:0    0   8G  0 disk
├─xvda1   202:1    0   8G  0 part /
├─xvda127 259:0    0   1M  0 part
└─xvda128 259:1    0  10M  0 part /boot/efi
[root@ip-10-0-0-203 ~]#
```

But beyond that, if you had data on this volume, you would want to make sure that it's still accessible.

Clean up resources:

To avoid unnecessary costs and maintain proper resource hygiene, delete the unencrypted volume and its snapshot:

1. **Navigate** to the **EC2 Console** → **Volumes**.
2. **Select** the unencrypted volume.
3. Click **Actions** → **Delete Volume**.
4. **Confirm** by typing **delete** and then click **Delete**.
5. Ensure the volume's **State** shows as **Available** (not **In-use**) before deletion.



Automating AWS EBS Volume Encryption using Terraform, Bash, and Checkov

Also automated the process of creating unencrypted AWS EBS volumes and EC2 instances using **Terraform**, then encrypts the unencrypted volumes using a **Bash script**, and finally scans the Terraform code for security misconfigurations using the **Checkov** tool.

Tools Used

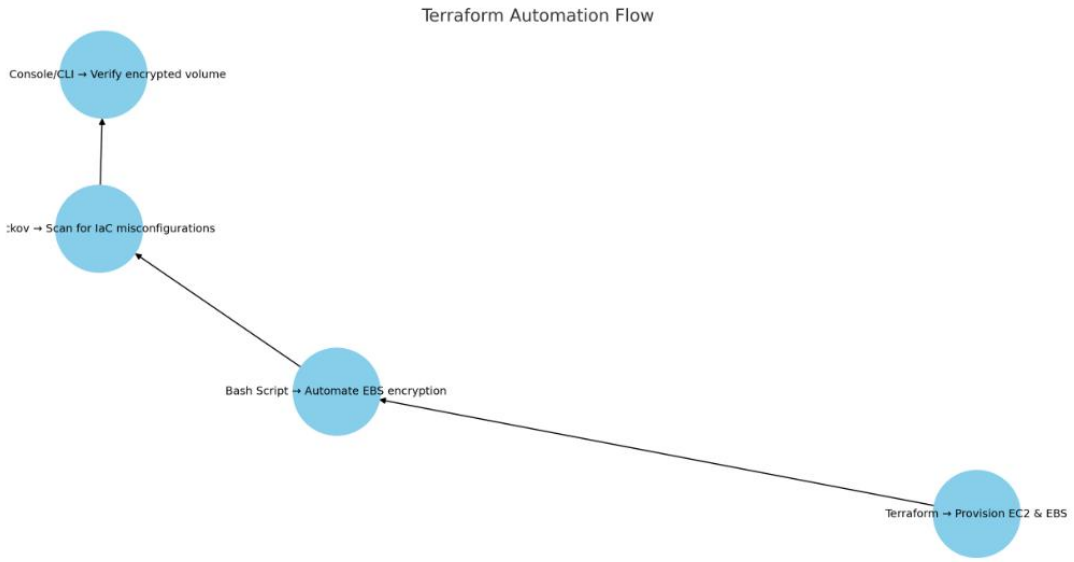
Terraform

Terraform is an open-source Infrastructure as Code (IaC) tool that enables users to define and provision infrastructure using a declarative configuration language. It simplifies cloud resource management and supports multi-cloud deployments.

Checkov

Checkov is a static code analysis tool for Infrastructure as Code (IaC). It scans Terraform, CloudFormation, Kubernetes, and other IaC frameworks for misconfigurations and security compliance violations.

Flow Chart:



Automating EBS Creation using Terraform

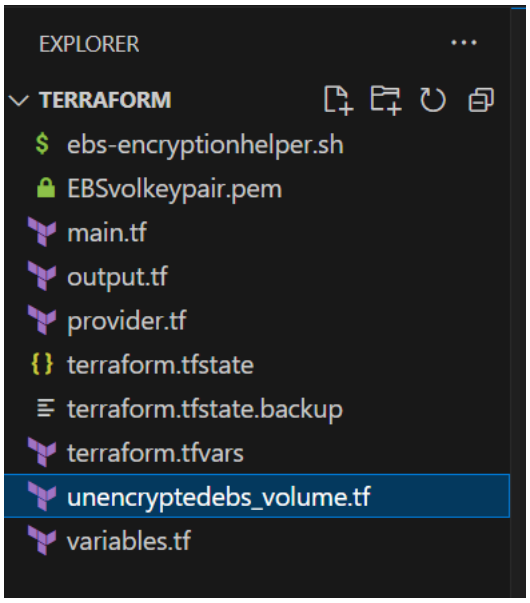
AWS Configuration

- **Run the following command to configure AWS credentials:**

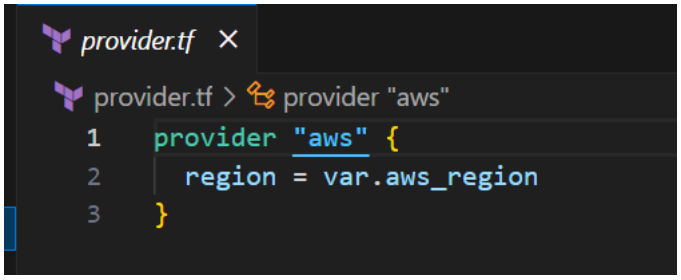
  aws configure

```
┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ aws configure
AWS Access Key ID [****************VRPT]:
AWS Secret Access Key [****************fgiw]:
Default region name [us-east-1]:
Default output format [json]:

┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ aws sts get-caller-identity
{
    "UserId": "
    "Account":
    "Arn": "
}
```

Write Terraform Code



- **provider.tf** → Defines AWS as the provider.



- **main.tf** → Contains resources for EC2 instance and unencrypted EBS volumes.

main.tf

```
terraform {
  required_version = ">= 1.3.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

resource "aws_instance" "ec2_instance" {
  ami           = var.ami_id
  instance_type = var.instance_type
  key_name      = var.key_name

  tags = {
    Name = "EBS-Encryption-Instance"
  }
}

resource "aws_ebs_volume" "unencrypted_volume" {
  availability_zone = var.availability_zone
  size              = var.volume_size
}

resource "aws_volume_attachment" "attach_volume" {
  device_name = "/dev/sdh"
  volume_id   = aws_ebs_volume.unencrypted_volume.id
  instance_id = aws_instance.ec2_instance.id
}
```

- **unencryptedebs_volume.tf** → Defines the unencrypted EBS volume.

unencryptedebs_volume.tf

unencryptedebs_volume.tf > resource "aws_ebs_volume" "unencrypted_volume1"

```
resource "aws_ebs_volume" "unencrypted_volume1" {
  availability_zone = var.availability_zone
  size              = var.volume_size
  encrypted         = false # ✅ Explicitly unencrypted

  tags = {
    Name = "Unencrypted-Volume1"
  }
}
```

- **variables.tf** → Manages input variables.

variables.tf

variables.tf > variable "aws_region"

```
variable "aws_region" {
  description = "AWS Region"
  default     = "us-east-1"
  type        = string
}

variable "availability_zone" {
  description = "The availability zone where the instance will be la
  type        = string
}

variable "instance_type" {
  description = "EC2 instance type"
  default     = "t2.micro"
}

variable "ami_id" {
  default     = "ami-05b10e08d247fb927"
  description = "AMI ID for EC2 instances"
  type        = string
}

variable "key_name" {
  description = "The name of the key pair"
  type        = string
}

variable "volume_size" {
  description = "The size of the EBS volume in GB"
  type        = number
}

variable "volume_type" {
  description = "The type of the EBS volume (e.g., gp2, gp3, io1)"
```

- **output.tf** → Outputs key resource information.

output.tf

output.tf > output "instance_id"

```
output "instance_id" {
  value = aws_instance.ec2_instance.id
}

output "volume_id" {
  value = aws_ebs_volume.unencrypted_volume.id
}
```

- **terraform.tfvars** → Holds variable values.

```
terraform.tfvars
aws_region          = "us-east-1"
ami_id              = "ami-05b10e08d247fb927"
instance_type       = "t2.micro"
key_name            = "EBSvolkeypair"
availability_zone   = "us-east-1b"
volume_size         = 8
volume_type         = "gp3"
```

Execute Terraform Commands

- terraform init # Initialize Terraform

```
terraform {
  required_version = ">= 1.3.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

resource "aws_instance" "ec2_instance" {
  ami           = var.ami_id

┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.88.0...
- Installed hashicorp/aws v5.88.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!
```

- terraform fmt # Format configuration files

```
┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ terraform fmt
unencryptedebs_volume.tf
variables.tf
```

- terraform validate # Validate the code

```
┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ terraform validate
Success! The configuration is valid.
```

- terraform plan # Preview the changes

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                                zsh

┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ terraform plan
aws_ebs_volume.unencrypted_volume: Refreshing state... [id=vol-03091415f153dfac7]
aws_instance.ec2_instance: Refreshing state... [id=i-0888aabdfc4a1d3bc]
aws_ebs_volume.unencrypted_volume1: Refreshing state... [id=vol-0dbd5d1b4351d4d64]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with
the following symbols:
  + create
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_ebs_volume.unencrypted_volume must be replaced
-/+ resource "aws_ebs_volume" "unencrypted_volume" {
      ~ arn                  = "arn:aws:ec2:us-east-1:337909744329:volume/vol-03091415f153dfac7" -> (known after app
```

- terraform apply # Apply the infrastructure

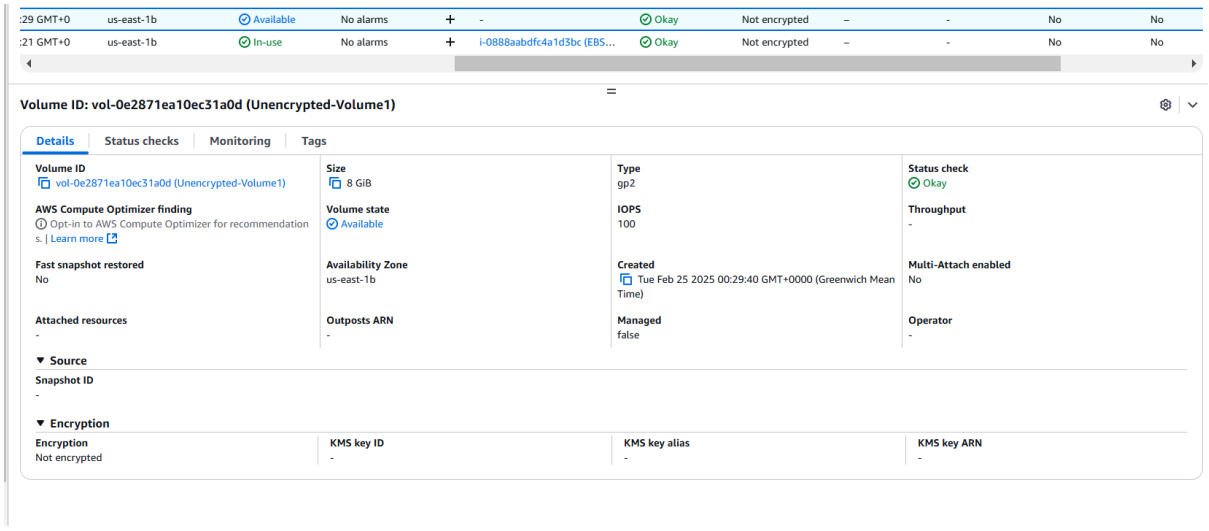
```
┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ terraform apply
aws_ebs_volume.unencrypted_volume1: Refreshing state... [id=vol-0dbd5d1b4351d4d64]
aws_ebs_volume.unencrypted_volume: Refreshing state... [id=vol-03091415f153dfac7]
aws_instance.ec2_instance: Refreshing state... [id=i-0888aabdfc4a1d3bc]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with
the following symbols:
  + create
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_ebs_volume.unencrypted_volume must be replaced
-/+ resource "aws_ebs_volume" "unencrypted_volume" {
      ~ arn                  = "arn:aws:ec2:us-east-1:337909744329:volume/vol-03091415f153dfac7" -> (known after app
```

```
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_ebs_volume.unencrypted_volume1: Destroying... [id=vol-0dbd5d1b4351d4d64]
aws_ebs_volume.unencrypted_volume: Destroying... [id=vol-03091415f153dfac7]
aws_ebs_volume.unencrypted_volume1: Still destroying... [id=vol-0dbd5d1b4351d4d64, 10s elapsed]
aws_ebs_volume.unencrypted_volume: Still destroying... [id=vol-03091415f153dfac7, 10s elapsed]
aws_ebs_volume.unencrypted_volume: Destruction complete after 11s
aws_ebs_volume.unencrypted_volume1: Destruction complete after 11s
aws_ebs_volume.unencrypted_volume: Creating...
aws_ebs_volume.unencrypted_volume1: Creating...
aws_ebs_volume.unencrypted_volume: Still creating... [10s elapsed]
aws_ebs_volume.unencrypted_volume1: Still creating... [10s elapsed]
aws_ebs_volume.unencrypted_volume: Creation complete after 11s [id=vol-0e35390d8b6e88a8c]
aws_volume_attachment.attach_volume: Creating...
aws_ebs_volume.unencrypted_volume1: Creation complete after 11s [id=vol-0e2871ea10ec31a0d]
aws_volume_attachment.attach_volume: Still creating... [10s elapsed]
aws_volume_attachment.attach_volume: Still creating... [20s elapsed]
aws_volume_attachment.attach_volume: Creation complete after 22s [id=vai-416811575]

Apply complete! Resources: 3 added, 0 changed, 2 destroyed.

Outputs:

instance_id = "i-0888aabdfc4a1d3bc"
volume_id = "vol-0e35390d8b6e88a8c"
```

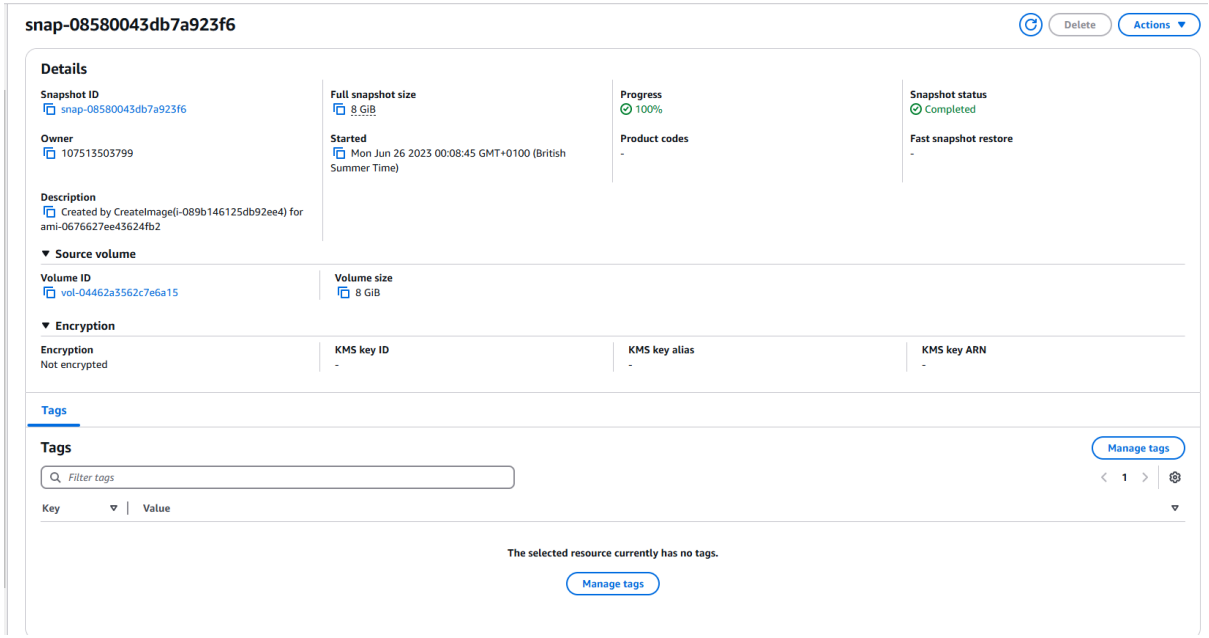
Deployment is successful, you can verify that manually by checking AWS management console:

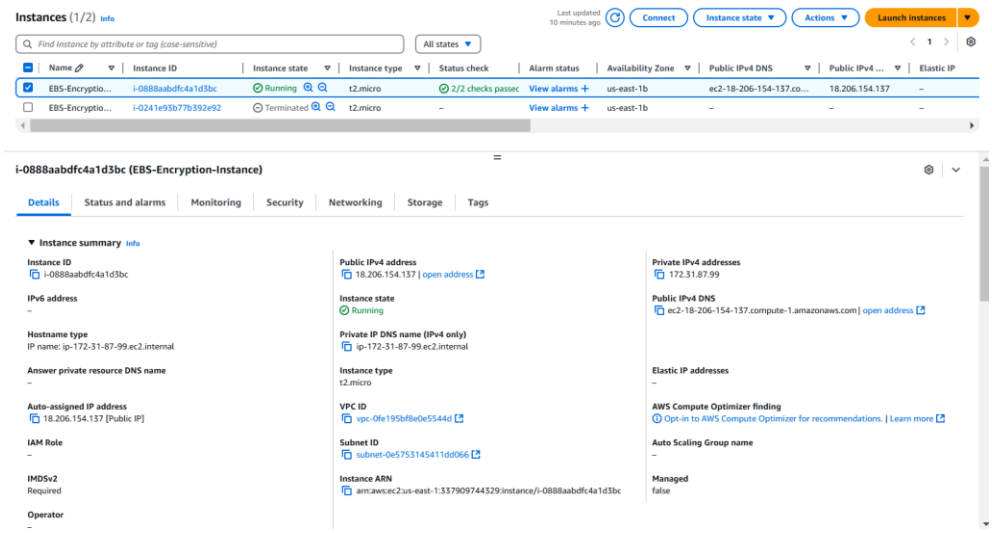
Verify EBS Creation

- AWS Management Console:
  - Check EBS volumes, snapshots, and EC2 instances.



- Check in snapshots





- Connecting to EC2 instance using SSH command as shown below:

```
┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ chmod 400 "EBSvolkeypair.pem"

┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ ssh -i "EBSvolkeypair.pem" ec2-user@ec2-18-206-154-137.compute-1.amazonaws.com
The authenticity of host 'ec2-18-206-154-137.compute-1.amazonaws.com (18.206.154.137)' can't be established.
ED25519 key fingerprint is SHA256:98dXF8+GEkUgNphbjRAJF6A56eoB3xqSPaiie6rHbRs.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-18-206-154-137.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
       ,     #_
       ~\_  ####_        Amazon Linux 2023
      ~~  \_#####\
      ~~     \###|
      ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
       ~~       V~' '->
        ~~~         /
          ~~._.   _/
             _/ _/
           _/m/'
[ec2-user@ip-172-31-87-99 ~]$ whoami
ec2-user
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

        ~~~         /
          ~~._.   _/
             _/ _/
           _/m/'
[ec2-user@ip-172-31-87-99 ~]$ whoami
ec2-user
[ec2-user@ip-172-31-87-99 ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  452K  190M   1% /run
/dev/xvda1      8.0G  1.6G  6.4G  20% /
tmpfs           475M     0  475M   0% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs            95M     0   95M   0% /run/user/1000
[ec2-user@ip-172-31-87-99 ~]$ lsblk
NAME      MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
xvda      202:0    0   8G  0 disk
├─xvda1   202:1    0   8G  0 part /
├─xvda127 259:0    0   1M  0 part
└─xvda128 259:1    0  10M  0 part /boot/efi
xvdh      202:112  0   8G  0 disk
[ec2-user@ip-172-31-87-99 ~]$
```

Using Terraform code successfully created unencrypted volumes.

Encrypting Unencrypted Volumes Using Bash Script

Run Bash Script

- The Bash script ebs-encryptionhelper.sh automates the encryption process.

EBSvolkeypair.pem $ ebs-encryptionhelper.sh

$ ebs-encryptionhelper.sh

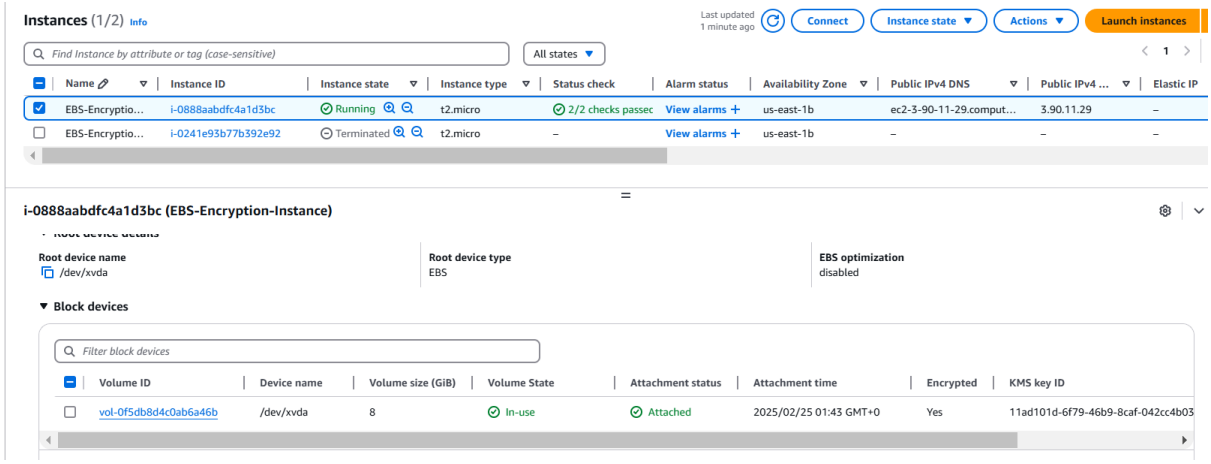
```bash
#!/bin/bash

REGION="us-east-1"
INSTANCE_ID="i-0888aabdfc4a1d3bc"  # Replace with your EC2 Instance ID
AVAILABILITY_ZONE="us-east-1b"  # Replace with your instance's AZ

# Stop the EC2 Instance
aws ec2 stop-instances --instance-ids $INSTANCE_ID --region $REGION
echo "Stopping instance $INSTANCE_ID..."
aws ec2 wait instance-stopped --instance-ids $INSTANCE_ID --region $REGION
echo "Instance $INSTANCE_ID stopped."

# Identify unencrypted EBS volumes
echo "Identifying unencrypted EBS volumes..."
unencrypted_volumes=$(aws ec2 describe-volumes --filters Name=attachment.instance-id,Values=$INSTANCE_ID -

for vol in $unencrypted_volumes; do
  echo "Processing volume: $vol"

  # Create Snapshot
  snapshot_id=$(aws ec2 create-snapshot --volume-id $vol --description "Snapshot of $vol for encryption" -
  echo "Snapshot created: $snapshot_id"

  # Wait for Snapshot Completion
  echo "Waiting for snapshot to complete..."
  aws ec2 wait snapshot-completed --snapshot-ids $snapshot_id --region $REGION

  # Copy Snapshot with Encryption
  encrypted_snapshot=$(aws ec2 copy-snapshot --source-region $REGION --source-snapshot-id $snapshot_id --e
  echo "Encrypted snapshot created: $encrypted_snapshot"

  # Wait for Encrypted Snapshot Completion
  echo "Waiting for encrypted snapshot to complete..."
  aws ec2 wait snapshot-completed --snapshot-ids $encrypted_snapshot --region $REGION
```

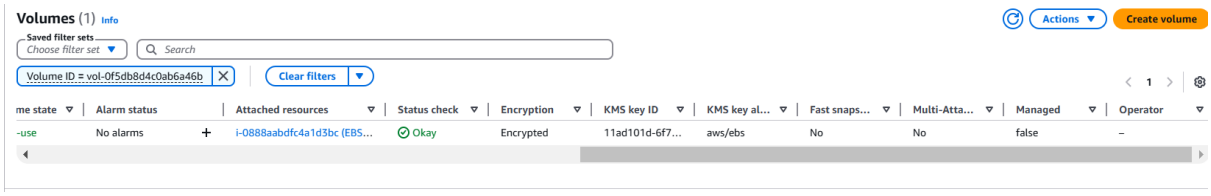
```
┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ ./ebs-encryptionhelper.sh
{
    "StoppingInstances": [
        {
            "CurrentState": {
                "Code": 64,
                "Name": "stopping"
            },
            "InstanceId": "i-0888aabdfc4a1d3bc",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
Stopping instance i-0888aabdfc4a1d3bc...
Instance i-0888aabdfc4a1d3bc stopped.
Identifying unencrypted EBS volumes...
Processing volume: vol-0467be32dfa110f3b
```

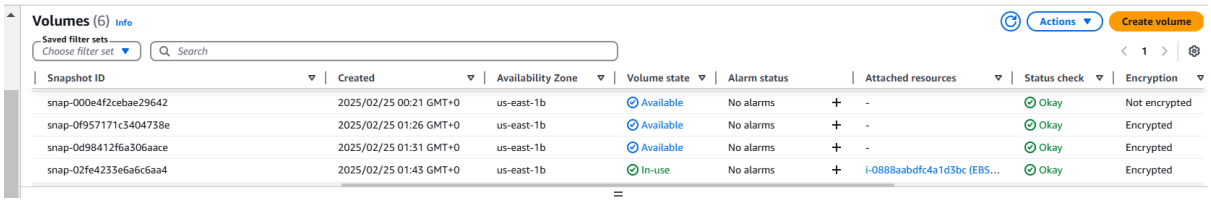
Ln 62, Col 1 Spaces: 4 UTF-8 LF Shell Script

Verify Encrypted Volumes

- **AWS Management Console:**
  - Check the updated EBS volume encryption status.







- **AWS CLI:**

```
┌──(kali㉿kali)-[~/EBSVolencryption]
aws ec2 describe-volumes --filters Name=attachment.instance-id,Values=i-0888aabdfc4a1d3bc --query "Volumes[*].{ID:VolumeId,Encrypted:Encrypted}" --region us-east-1 --output table
| Encrypted |          ID          |
+-----------+----------------------+
| True      | vol-0f5db8d4c0ab6a46b |
+-----------+----------------------+
```

Try connecting to EC2 instance, to verify the volumes created or not using SSH

```
┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ ssh -i "EBSvolkeypair.pem" ec2-user@ec2-3-90-11-29.compute-1.amazonaws.com
The authenticity of host 'ec2-3-90-11-29.compute-1.amazonaws.com (3.90.11.29)' can't be established.
ED25519 key fingerprint is SHA256:98dXF8+GEkUgNphbjRAJF6A56eoB3xqSPaiie6rHbRs.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:6: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-90-11-29.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Tue Feb 25 01:02:34 2025 from 90.251.127.120
[ec2-user@ip-172-31-87-99 ~]$ whoami
ec2-user
[ec2-user@ip-172-31-87-99 ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  448K  190M   1% /run
/dev/xvda1      8.0G  1.6G  6.4G  21% /
tmpfs           475M     0  475M   0% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
```

```
tmpfs           95M     0   95M   0% /run/user/1000
[ec2-user@ip-172-31-87-99 ~]$ lsblk
NAME      MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
xvda      202:0    0   8G  0 disk
├─xvda1   202:1    0   8G  0 part /
├─xvda127 259:0    0   1M  0 part
└─xvda128 259:1    0  10M  0 part /boot/efi
```

Scanning Terraform Code Using Checkov

- **Install Checkov**
  pip install checkov
- **Run Checkov Scan**
  checkov -d /home/kali/EBSVolencryption/

```
┌──(checkov-env)─(kali㉿kali)-[~/EBSVolencryption]
└─$ checkov -d /home/kali/EBSVolencryption/

2025-02-24 21:29:59,188 [MainThread  ] [WARNI]  An unsupported instruction IMPORT was used in /checkov-env/lib/pytho
n3.13/site-packages/checkov/common/util/dockerfile.py
2025-02-24 21:29:59,194 [MainThread  ] [WARNI]  An unsupported instruction DOCKERFILE_MASK was used in /checkov-env/
lib/python3.13/site-packages/checkov/common/util/dockerfile.py
2025-02-24 21:29:59,199 [MainThread  ] [WARNI]  An unsupported instruction DEF was used in /checkov-env/lib/python3.
13/site-packages/checkov/common/util/dockerfile.py
2025-02-24 21:29:59,204 [MainThread  ] [WARNI]  An unsupported instruction IF was used in /checkov-env/lib/python3.1
3/site-packages/checkov/common/util/dockerfile.py
2025-02-24 21:29:59,205 [MainThread  ] [WARNI]  An unsupported instruction RETURN was used in /checkov-env/lib/pytho
n3.13/site-packages/checkov/common/util/dockerfile.py
2025-02-24 21:29:59,214 [MainThread  ] [WARNI]  An unsupported instruction RETURN was used in /checkov-env/lib/pytho
n3.13/site-packages/checkov/common/util/dockerfile.py
[ dockerfile framework ]: 100%|[1/1], Current File Scanned=../../../checkov-env/lib/python3.13/
[ terraform framework ]: 100%|[5/5], Current File Scanned=variables.tfv-env/lib/python3.13/site
[ kubernetes framework ]: 100%|[1349/1349], Current File Scanned=checkov-env/lib/python3.13/sit
[ secrets framework ]:   1%|[15/1351], Current File Scanned=/home/kali/EBSVolencryption/checkov
```

```
[ kubernetes framework ]: 100%|                    |[1349/1349], Current File Scanned=checkov-env/lib/python3.13/sit
[ secrets framework ]: 100%|                    |[1351/1351], Current File Scanned=/home/kali/EBSVolencryption/check

       _                _
   ___| |__   ___  ___| | _______   __
  / __| '_ \ / _ \/ __| |/ / _ \ \ / /
 | (__| | | |  __/ (__|   < (_) \ V /
  \___|_| |_|\___|\___|_|\_\___/ \_/

By Prisma Cloud | version: 3.2.373

terraform scan results:

Passed checks: 4, Failed checks: 10, Skipped checks: 0

Check: CKV_AWS_46: "Ensure no hard-coded secrets exist in EC2 user data"
        PASSED for resource: aws_instance.ec2_instance
        File: /main.tf:11-19
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/secrets-policies/bc-a
ws-secrets-1
Check: CKV_AWS_88: "EC2 instance should not have public IP."
        PASSED for resource: aws_instance.ec2_instance
        File: /main.tf:11-19
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/public-policies/publi
c-12
Check: CKV_AWS_41: "Ensure no hard coded AWS access key and secret key exists in provider"
        PASSED for resource: aws.default
```

Review security findings and address misconfigurations.

Clean Up Resources

- Destroy all resources created by Terraform:

  terraform destroy

Performance Impact of EBS Encryption

While AWS EBS encryption is designed to have minimal impact on performance, specific workloads may experience slight variances in latency or IOPS. Using EBS-optimized instances ensures consistent and high performance.

Considerations:

- **Instance Types:** Select EBS-optimized instances for workloads requiring high IOPS.
- **IOPS and Throughput:** Encrypted volumes support the same IOPS and throughput as unencrypted ones.
- **Benchmarking:** Use tools like fio to measure performance impacts in your specific environment.

Compliance and Security Best Practices

EBS encryption supports compliance with regulations like HIPAA, PCI DSS, and GDPR. Following AWS security best practices ensures robust data protection.

- **Best Practices:**
  - Implement least privilege access using AWS IAM policies.
  - Enable CloudTrail logging for monitoring KMS key usage.
  - Regularly rotate encryption keys and use CMKs for sensitive data.

    - Apply AWS Config rules to enforce encryption compliance.
    - Conduct periodic audits and vulnerability assessments.

Incident Response and Data Recovery

A comprehensive data recovery plan ensures data integrity in case of accidental deletions or breaches. Using AWS Backup and automated snapshots helps maintain up-to-date data copies.

- **Key Points:**
    - Schedule regular snapshots using AWS Data Lifecycle Manager.
    - Implement cross-region backups for disaster recovery.
    - Use AWS Backup for centralized backup management across services.
    - Configure CloudWatch alarms for snapshot failures.
    - Test recovery procedures regularly to ensure data availability.

Additional Tips

- **Enable EBS Encryption by Default:** Set this at the account level if all EBS volumes should be encrypted. Navigate to **EC2 Dashboard** → **Account Attributes** → **Data Protection and Security** → **EBS Encryption** and set **"Always encrypt new EBS volumes"** to **"Enabled"**.
- **Backup Regularly:** Regularly create snapshots of your volumes to ensure you have up-to-date backups in case of data loss or corruption. *(Automate this process for efficiency.)*
- **Automate Snapshot Management:** Use **AWS Data Lifecycle Manager** to automate the creation and retention of EBS snapshots, eliminating manual interventions.
- **Monitor Volume Usage:** Utilize **AWS CloudWatch** to track EBS volume performance and usage metrics, helping to identify potential issues early.
- **Cost Management:** Periodically review EBS usage and remove unused volumes or outdated snapshots to optimize costs.
- **Tagging:** Tag all resources (volumes, snapshots) for better organization, cost allocation, and easier identification of orphaned resources.
- **Enforce Encryption with AWS Config:** Besides enabling encryption by default, use **AWS Config** to enforce policies ensuring all new EBS volumes remain encrypted.
- **Instance Metadata Service (IMDS):** Ensure **IMDS** is enabled and securely configured to allow instances to retrieve necessary credentials for encryption and other operations.

Real-World Use Cases and Scenarios

- **Financial Services:** Encrypting sensitive transaction data to comply with PCI DSS.
- **Healthcare Organizations:** Ensuring HIPAA compliance by encrypting patient data.
- **Startups:** Using customer-managed keys for enhanced security and compliance.
- **E-commerce Platforms:** Protecting customer payment data and personal information.
- **Government Agencies:** Meeting strict data sovereignty and security requirements.

Conclusion

Securing Amazon EBS volumes with encryption is a vital step in protecting sensitive data within AWS environments. By leveraging AWS Key Management Service (KMS) for key control, implementing fine-grained IAM policies, and using security tools like Checkov for infrastructure compliance, organizations can significantly reduce the risk of data breaches and unauthorized access.

However, encryption alone is not a silver bullet. It must be part of a broader security strategy that includes continuous monitoring, regular backups, automated snapshot management, and performance tracking. Integrating Infrastructure as Code (IaC) practices with Terraform and incorporating security checks ensures consistency, scalability, and proactive risk mitigation.

By adopting this complete framework for EBS data protection and encryption, businesses can not only meet stringent compliance requirements like HIPAA, PCI DSS, and GDPR but also strengthen their overall cloud security posture. This layered approach fosters operational resilience, enhances data integrity, and builds trust with clients and stakeholders in an increasingly cloud-centric world.

References:

AWS. (n.d.). *Amazon EC2 instance connect prerequisites*. Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-connect-prerequisites.html [Accessed 25 Feb. 2025].

AWS. (n.d.). *Amazon EC2 instance storage: EBS-optimized instances*. Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-optimized.html [Accessed 25 Feb. 2025].

AWS. (n.d.). *AWS Backup Developer Guide*. Available at: https://docs.aws.amazon.com/aws-backup/latest/devguide/whatisbackup.html [Accessed 25 Feb. 2025].

AWS. (n.d.). *AWS compliance programs*. Available at: https://aws.amazon.com/compliance/programs/ [Accessed 25 Feb. 2025].

AWS. (n.d.). *AWS Key Management Service (KMS) Developer Guide*. Available at: https://docs.aws.amazon.com/kms/latest/developerguide/overview.html [Accessed 25 Feb. 2025].

AWS. (n.d.). *AWS Well-Architected Framework: Security Pillar*. Available at: https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/welcome.html [Accessed 25 Feb. 2025].

AWS. (n.d.). *Amazon EBS Encryption*. Available at: https://docs.aws.amazon.com/ebs/latest/userguide/ebs-encryption.html [Accessed 25 Feb. 2025].

AWS. (n.d.). *Amazon Elastic Block Store (EBS) volumes*. Available at: https://docs.aws.amazon.com/ebs/latest/userguide/ebs-volumes.html [Accessed 25 Feb. 2025].

Checkov. (n.d.). *Installing Checkov*. Available at: https://www.checkov.io/2.Basics/Installing%20Checkov.html [Accessed 25 Feb. 2025].

Cybr. (n.d.). *EBS Data Protection and Encryption*. Available at: https://cybr.com/courses/introduction-to-aws-security/lessons/ebs-data-protection-and-encryption/ [Accessed 25 Feb. 2025].

Terraform. (n.d.). *AWS Provider Documentation*. Available at: https://registry.terraform.io/providers/hashicorp/aws/latest/docs [Accessed 25 Feb. 2025].

Appendices:

```
┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ terraform plan
aws_ebs_volume.unencrypted_volume: Refreshing state... [id=vol-03091415f153dfac7]
aws_instance.ec2_instance: Refreshing state... [id=i-0888aabdfc4a1d3bc]
aws_ebs_volume.unencrypted_volume1: Refreshing state... [id=vol-0dbd5d1b4351d4d64]


Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with
the following symbols:
  + create
-/+ destroy and then create replacement


Terraform will perform the following actions:


  # aws_ebs_volume.unencrypted_volume must be replaced
-/+ resource "aws_ebs_volume" "unencrypted_volume" {
      ~ arn                          = "arn:aws:ec2:us-east-1:337909744329:volume/vol-03091415f153dfac7" -> (known after apply)
      ~ availability_zone     = "us-east-1a" -> "us-east-1b" # forces replacement
      ~ encrypted            = false -> (known after apply)
      ~ id                  = "vol-03091415f153dfac7" -> (known after apply)
      ~ iops                 = 100 -> (known after apply)
      + kms_key_id           = (known after apply)
      - multi_attach_enabled = false -> null
      + snapshot_id          = (known after apply)
      - tags                 = {} -> null
```

```
      ~ tags_all             = {} -> (known after apply)
      ~ throughput           = 0 -> (known after apply)
      ~ type                 = "gp2" -> (known after apply)
        # (2 unchanged attributes hidden)
    }

  # aws_ebs_volume.unencrypted_volume1 must be replaced
-/+ resource "aws_ebs_volume" "unencrypted_volume1" {
      ~ arn                          = "arn:aws:ec2:us-east-1:337909744329:volume/vol-0dbd5d1b4351d4d64" -> (known after apply)
      ~ availability_zone     = "us-east-1a" -> "us-east-1b" # forces replacement
      ~ id                  = "vol-0dbd5d1b4351d4d64" -> (known after apply)
      ~ iops                 = 100 -> (known after apply)
      + kms_key_id           = (known after apply)
      - multi_attach_enabled = false -> null
      + snapshot_id          = (known after apply)
        tags                 = {
            "Name" = "Unencrypted-Volume1"
        }
      ~ throughput           = 0 -> (known after apply)
      ~ type                 = "gp2" -> (known after apply)
        # (4 unchanged attributes hidden)
    }

  # aws_volume_attachment.attach_volume will be created
  + resource "aws_volume_attachment" "attach_volume" {
      + device_name = "/dev/sdh"
      + id          = (known after apply)
      + instance_id = "i-0888aabdfc4a1d3bc"
```

```
      + volume_id   = (known after apply)
    }

Plan: 3 to add, 0 to change, 2 to destroy.

Changes to Outputs:
  ~ volume_id  = "vol-03091415f153dfac7" -> (known after apply)

─────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions
if you run "terraform apply" now.

┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ terraform apply
aws_ebs_volume.unencrypted_volume1: Refreshing state... [id=vol-0dbd5d1b4351d4d64]
aws_ebs_volume.unencrypted_volume: Refreshing state... [id=vol-03091415f153dfac7]
aws_instance.ec2_instance: Refreshing state... [id=i-0888aabdfc4a1d3bc]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with
the following symbols:
  + create
-/+ destroy and then create replacement

Terraform will perform the following actions:
```

```
  # aws_ebs_volume.unencrypted_volume must be replaced
-/+ resource "aws_ebs_volume" "unencrypted_volume" {
      ~ arn                          = "arn:aws:ec2:us-east-1:337909744329:volume/vol-03091415f153dfac7" -> (known after apply)
      ~ availability_zone     = "us-east-1a" -> "us-east-1b" # forces replacement
      ~ encrypted            = false -> (known after apply)
      ~ id                  = "vol-03091415f153dfac7" -> (known after apply)
      ~ iops                 = 100 -> (known after apply)
      + kms_key_id           = (known after apply)
      - multi_attach_enabled = false -> null
      + snapshot_id          = (known after apply)
      - tags                 = {} -> null
      ~ tags_all             = {} -> (known after apply)
      ~ throughput           = 0 -> (known after apply)
      ~ type                 = "gp2" -> (known after apply)
        # (2 unchanged attributes hidden)
    }


  # aws_ebs_volume.unencrypted_volume1 must be replaced
-/+ resource "aws_ebs_volume" "unencrypted_volume1" {
      ~ arn                          = "arn:aws:ec2:us-east-1:337909744329:volume/vol-0dbd5d1b4351d4d64" -> (known after apply)
      ~ availability_zone     = "us-east-1a" -> "us-east-1b" # forces replacement
      ~ id                  = "vol-0dbd5d1b4351d4d64" -> (known after apply)
      ~ iops                 = 100 -> (known after apply)
      + kms_key_id           = (known after apply)
      - multi_attach_enabled = false -> null
      + snapshot_id          = (known after apply)
```

```
        tags                 = {
            "Name" = "Unencrypted-Volume1"
        }
      ~ throughput           = 0 -> (known after apply)
      ~ type                 = "gp2" -> (known after apply)
        # (4 unchanged attributes hidden)
    }

  # aws_volume_attachment.attach_volume will be created
  + resource "aws_volume_attachment" "attach_volume" {
      + device_name = "/dev/sdh"
      + id          = (known after apply)
      + instance_id = "i-0888aabdfc4a1d3bc"
      + volume_id   = (known after apply)
    }

Plan: 3 to add, 0 to change, 2 to destroy.

Changes to Outputs:
  ~ volume_id   = "vol-03091415f153dfac7" -> (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_ebs_volume.unencrypted_volume1:   Destroying...   [id=vol-0dbd5d1b4351d4d64]
aws_ebs_volume.unencrypted_volume:   Destroying...   [id=vol-03091415f153dfac7]
```

```
aws_ebs_volume.unencrypted_volume1: Still destroying... [id=vol-0dbd5d1b4351d4d64, 10s elapsed]

aws_ebs_volume.unencrypted_volume: Still destroying... [id=vol-03091415f153dfac7, 10s elapsed]

aws_ebs_volume.unencrypted_volume: Destruction complete after 11s

aws_ebs_volume.unencrypted_volume1: Destruction complete after 11s

aws_ebs_volume.unencrypted_volume: Creating...

aws_ebs_volume.unencrypted_volume1: Creating...

aws_ebs_volume.unencrypted_volume: Still creating... [10s elapsed]

aws_ebs_volume.unencrypted_volume1: Still creating... [10s elapsed]

aws_ebs_volume.unencrypted_volume: Creation complete after 11s [id=vol-0e35390d8b6e88a8c]

aws_volume_attachment.attach_volume: Creating...

aws_ebs_volume.unencrypted_volume1: Creation complete after 11s [id=vol-0e2871ea10ec31a0d]

aws_volume_attachment.attach_volume: Still creating... [10s elapsed]

aws_volume_attachment.attach_volume: Still creating... [20s elapsed]

aws_volume_attachment.attach_volume: Creation complete after 22s [id=vai-416811575]


Apply complete! Resources: 3 added, 0 changed, 2 destroyed.


Outputs:


instance_id = "i-0888aabdfc4a1d3bc"

volume_id = "vol-0e35390d8b6e88a8c"


┌──(kali㉿kali)-[~/EBSVolencryption]
```

└─$

┌──(kali㉿kali)-[~/EBSVolencryption]

└─$ chmod 400 "EBSvolkeypair.pem"

┌──(kali㉿kali)-[~/EBSVolencryption]

└─$ ssh -i "EBSvolkeypair.pem" ec2-user@ec2-18-206-154-137.compute-1.amazonaws.com

The authenticity of host 'ec2-18-206-154-137.compute-1.amazonaws.com (18.206.154.137)' can't be established.

ED25519 key fingerprint is SHA256:98dXF8+GEkUgNphbjRAJF6A56eoB3xqSPaiie6rHbRs.

This key is not known by any other names.

Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

Warning: Permanently added 'ec2-18-206-154-137.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
```

[ec2-user@ip-172-31-87-99 ~]$ whoami ec2-user [ec2-user@ip-172-31-87-99 ~]$ df -h Filesystem Size Used Avail Use% Mounted on devtmpfs 4.0M 0 4.0M 0% /dev tmpfs 475M 0 475M 0% /dev/shm tmpfs 190M 452K 190M 1% /run /dev/xvda1 8.0G 1.6G 6.4G 20% / tmpfs 475M 0 475M 0% /tmp /dev/xvda128 10M 1.3M 8.7M 13% /boot/efi tmpfs 95M 0 95M 0% /run/user/1000 [ec2-user@ip-172-31-87-99 ~]$ lsblk NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS xvda 202:0 0 8G 0 disk ├─xvda1 202:1 0 8G 0 part / ├─xvda127 259:0 0 1M 0 part └─xvda128 259:1 0 10M 0 part /boot/efi xvdh 202:112 0 8G 0 di

```
┌──(kali㉿kali)-[~/EBSVolencryption]
└─$ ./ebs-encryptionhelper.sh
{
    "StoppingInstances": [
        {
            "CurrentState": {
                "Code": 64,
                "Name": "stopping"
            },
            "InstanceId": "i-0888aabdfc4a1d3bc",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
Stopping instance i-0888aabdfc4a1d3bc...
Instance i-0888aabdfc4a1d3bc stopped.
Identifying unencrypted EBS volumes...
Processing volume: vol-0467be32dfa110f3b
Snapshot created: snap-079b69e791ed0145c
Waiting for snapshot to complete...
Encrypted snapshot created: snap-02fe4233e6a6c6aa4
Waiting for encrypted snapshot to complete...
Encrypted volume created: vol-0f5db8d4c0ab6a46b
Waiting for encrypted volume to become available...
{
    "AttachTime": "2025-02-25T00:21:55+00:00",
```

```
    "Device": "/dev/xvda",
    "InstanceId": "i-0888aabdfc4a1d3bc",
    "State": "detaching",
    "VolumeId": "vol-0467be32dfa110f3b"
}
```


Detached unencrypted volume: vol-0467be32dfa110f3b


```
{
    "AttachTime": "2025-02-25T01:43:40.838000+00:00",
    "Device": "/dev/xvda",
    "InstanceId": "i-0888aabdfc4a1d3bc",
    "State": "attaching",
    "VolumeId": "vol-0f5db8d4c0ab6a46b"
}
```


Encrypted volume vol-0f5db8d4c0ab6a46b attached to instance i-0888aabdfc4a1d3bc.


```
{
    "StartingInstances": [
        {
            "CurrentState": {
                "Code": 0,
                "Name": "pending"
            },
            "InstanceId": "i-0888aabdfc4a1d3bc",
            "PreviousState": {
                "Code": 80,
                "Name": "stopped"
            }
        }
    ]
}
```


Starting instance i-0888aabdfc4a1d3bc...

Instance i-0888aabdfc4a1d3bc is running.

┌──(kali㉿kali)-[~/EBSVolencryption] └─$ ssh -i "EBSvolkeypair.pem" ec2-user@ec2-3-90-11-29.compute-1.amazonaws.com The authenticity of host 'ec2-3-90-11-29.compute-1.amazonaws.com (3.90.11.29)' can't be established. ED25519 key fingerprint is SHA256:98dXF8+GEkUgNphbjRAJF6A56eoB3xqSPaiie6rHbRs. This host key is known by the following other names/addresses: ~/.ssh/known_hosts:6: [hashed name] Are you sure you want to continue connecting (yes/no/[fingerprint])? yes Warning: Permanently added 'ec2-3-90-11-29.compute-1.amazonaws.com' (ED25519) to the list of known hosts. , #_ ~_ ####_ Amazon Linux 2023 ~~ _#####\ ~~ \###| ~~ \#/ ___ https://aws.amazon.com/linux/amazon-linux-2023 ~~ V~' '-> ~~~ / ~~._. _/ _/ _/ _/m/' Last login: Tue Feb 25 01:02:34 2025 from 90.251.127.120 [ec2-user@ip-172-31-87-99 ~]$ whoami ec2-user [ec2-user@ip-172-31-87-99 ~]$ df -h Filesystem Size Used Avail Use% Mounted on devtmpfs 4.0M 0 4.0M 0% /dev tmpfs 475M 0 475M 0% /dev/shm tmpfs 190M 448K 190M 1% /run /dev/xvda1 8.0G 1.6G 6.4G 21% / tmpfs 475M 0 475M 0% /tmp /dev/xvda128 10M 1.3M 8.7M 13% /boot/efi tmpfs 95M 0 95M 0% /run/user/1000 [ec2-user@ip-172-31-87-99 ~]$ lsblk NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS xvda 202:0 0 8G 0 disk ├─xvda1 202:1 0 8G 0 part / ├─xvda127 259:0 0 1M 0 part └─xvda128 259:1 0 10M 0 part /boot/efi [ec2-user@ip-172-31-87-99 ~]$ exit logout Connection to ec2-3-90-11-29.compute-1.amazonaws.com closed. ┌──(kali㉿kali)-[~/EBSVolencryption] └─$ aws ec2 describe-volumes --filters Name=attachment.instance-id,Values=i-0888aabdfc4a1d3bc --query "Volumes[*].{ID:VolumeId,Encrypted:Encrypted}" --region us-east-1 --output table ---------- | DescribeVolumes | +----------+----------+ | Encrypted | ID | +----------+----------+ | True | vol-0f5db8d4c0ab6a46b | +----------+----------+