Automating ENI Failover with AWS Lambda + EventBridge

Goal: Automatically detach one or more Elastic Network Interfaces (ENIs) from a primary EC2 instance and attach them to a secondary EC2 instance when the primary transitions to a stopped/terminated state. This guide walks you through the setup **entirely from the AWS Management Console**.

Architecture overview

1. **EventBridge** receives EC2 Instance State-change Notification for the *primary* instance (e.g. `stopped`, `terminated`).
2. EventBridge triggers the **Lambda** function.
3. **Lambda** calls EC2 APIs to detach the configured ENIs from the primary instance and attach them to the secondary instance.

Prerequisites

- Primary and secondary EC2 instance IDs.
- ENI IDs you want to move (ENIs must be in the *same Availability Zone* as the target instance).
- IAM permissions to create roles, Lambda, and EventBridge rules.

Step 1 — Create the IAM role for Lambda

1. Go to the **IAM Console** → **Roles** → **Create role**.
2. Select **Trusted entity type: AWS service** → **Use case: Lambda** → **Next**.
3. Attach the managed policy **AWSLambdaBasicExecutionRole** (for logging).
4. Click **Next**.

5. Name the role `lambda-eni-mover-role` and create it.
6. After creation, open the role and go to the **Permissions** tab → **Add permissions** → **Create inline policy**.
    a. Choose **JSON** editor and paste:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DetachNetworkInterface",
        "ec2:AttachNetworkInterface"
      ],
      "Resource": "*"
    }
  ]
}
```

7. Save the policy with name `LambdaEC2ENIPermissions`.

Step 2 — Create the Lambda function

1. Go to the **Lambda Console** → **Create function**.
2. Choose **Author from scratch**.
    a. Function name: `eni-mover`
    b. Runtime: **Python 3.10**
    c. Execution role: Choose **Use existing role** and pick `lambda-eni-mover-role`.
3. Click **Create function**.
4. Once created, scroll down to the **Code source** editor. Replace the default code with the provided Python code (see below).
5. In the **Configuration** tab → **General configuration**, set:
    a. Timeout: **3 minutes (180 seconds)**

b. Memory: **512 MB** (adjustable)

6. In the **Configuration** tab → **Environment variables**, add:
    a. `PRIMARY_INSTANCE` = your primary instance ID
    b. `SECONDARY_INSTANCE` = your secondary instance ID
    c. `SECONDARY_ENIS` = comma-separated ENI IDs (e.g., `eni-abc123,eni-def456`)

Lambda function code (paste into editor):

```
import boto3
import time
from concurrent.futures import ThreadPoolExecutor, as_completed

ec2 = boto3.client('ec2')

PRIMARY_INSTANCE = "INSTANCE_ID_1"
SECONDARY_INSTANCE = "INSTANCE_ID_2"
SECONDARY_ENIS = [
    "ENI_ID_1",
    "ENI_ID_2"
]

def wait_for_available(eni_id, timeout=180):
    start = time.time()
    while time.time() - start < timeout:
        eni =
ec2.describe_network_interfaces(NetworkInterfaceIds=[eni_id])['NetworkInterfaces'][0]
        if eni['Status'] == 'available':
            print(f"ENI {eni_id} is now available.")
            return True
        time.sleep(3)
    raise TimeoutError(f"ENI {eni_id} did not become available in {timeout} seconds")

def wait_for_in_use(eni_id, timeout=180):
    start = time.time()
    while time.time() - start < timeout:
        eni =
ec2.describe_network_interfaces(NetworkInterfaceIds=[eni_id])['NetworkInterfaces'][0]
        if eni['Status'] == 'in-use':
            print(f"ENI {eni_id} is now attached.")
            return True
        time.sleep(3)
    raise TimeoutError(f"ENI {eni_id} did not attach in {timeout} seconds")
```

```
def move_eni_to_secondary(eni_id, device_index):
    eni =
ec2.describe_network_interfaces(NetworkInterfaceIds=[eni_id])['NetworkInterfaces'][0]
    attachment = eni.get('Attachment')

    if attachment:
        print(f"Detaching ENI {eni_id} from {attachment['InstanceId']}...")
        ec2.detach_network_interface(AttachmentId=attachment['AttachmentId'],
Force=True)
        wait_for_available(eni_id)
    else:
        print(f"ENI {eni_id} already detached.")

    print(f"Attaching ENI {eni_id} to secondary at DeviceIndex {device_index}...")
    ec2.attach_network_interface(NetworkInterfaceId=eni_id,
InstanceId=SECONDARY_INSTANCE, DeviceIndex=device_index)
    wait_for_in_use(eni_id)
    return eni_id

def lambda_handler(event, context):
    print("Event received:", event)

    detail = event.get("detail", {})
    if detail.get("instance-id") != PRIMARY_INSTANCE:
        print("Event not for primary instance. Skipping.")
        return {"status": "skipped - not primary"}
    if detail.get("state") not in ["stopping", "stopped", "shutting-down",
"terminated"]:
        print("Instance state not relevant. Skipping.")
        return {"status": "skipped - irrelevant state"}

    # Collect current device indices on secondary
    existing_indexes = []
    response = ec2.describe_instances(InstanceIds=[SECONDARY_INSTANCE])
    for iface in response['Reservations'][0]['Instances'][0]['NetworkInterfaces']:
        existing_indexes.append(iface['Attachment']['DeviceIndex'])

    # Prepare device indices for each ENI
    device_indices = []
    next_index = 1
    for _ in SECONDARY_ENIS:
        while next_index in existing_indexes:
            next_index += 1
        device_indices.append(next_index)
```

```
        existing_indexes.append(next_index)
        next_index += 1

    # Move ENIs in parallel
    results = []
    with ThreadPoolExecutor(max_workers=len(SECONDARY_ENIS)) as executor:
        future_to_eni = {executor.submit(move_eni_to_secondary, eni, idx): eni for
eni, idx in zip(SECONDARY_ENIS, device_indices)}
        for future in as_completed(future_to_eni):
            eni_id = future_to_eni[future]
            try:
                result = future.result()
                print(f"{result} moved successfully.")
                results.append(result)
            except Exception as e:
                print(f"Error moving {eni_id}: {e}")

    return {"status": "ENI move complete", "moved_enis": results}
```

Click **Deploy**.

Step 3 — Create the EventBridge rule

1. Go to **Amazon EventBridge Console** → **Rules** → **Create rule**.
2. Name: `eni-mover-primary-stop`.
3. Rule type: **Rule with event pattern**.
4. Event pattern → **Custom pattern (JSON editor)** → paste:

```
{
  "source": ["aws.ec2"],
  "detail-type": ["EC2 Instance State-change Notification"],
  "detail": {
    "instance-id": ["PRIMARY_INSTANCE_ID"],
    "state": ["stopped", "terminated"]
  }
}
```

5. Next, add a **Target** → choose **Lambda function** → select `eni-mover`.
6. Create the rule.

EventBridge will now automatically invoke the Lambda when the primary instance transitions to `stopped` or `terminated`.

Step 4 — Testing the setup

Option A: Test manually in Lambda console

1. Open your Lambda function.
2. Go to the **Test** tab.
3. Create a new test event, paste this sample payload:

```
{
  "version": "0",
  "id": "abcd-1234",
  "detail-type": "EC2 Instance State-change Notification",
  "source": "aws.ec2",
  "account": "111122223333",
  "time": "2025-09-16T12:00:00Z",
  "region": "us-east-1",
  "resources": ["arn:aws:ec2:us-east-
1:111122223333:instance/PRIMARY_INSTANCE_ID"],
  "detail": {
    "instance-id": "PRIMARY_INSTANCE_ID",
    "state": "stopped"
  }
}
```

4. Run the test and check **Monitor → Logs** for details.

Option B: Stop the primary instance

- In the **EC2 Console**, stop your primary instance.
- Monitor the Lambda logs in **CloudWatch Logs** to verify ENIs were detached and reattached.

Troubleshooting tips

- **AZ mismatch:** ENI and target instance must be in the same Availability Zone.
- **Device index errors:** Lambda code auto-assigns free indices > 0, but if conflicts occur, check existing attachments.
- **No trigger:** Ensure EventBridge rule's `instance-id` exactly matches the Primary instance ID.
- **Permission errors:** Verify `lambda-eni-mover-role` has the inline policy applied.

Optional enhancements

- Use **Systems Manager Parameter Store** to store ENI IDs instead of environment variables.
- Add **SNS notifications** in Lambda for success/failure.
- Add **CloudWatch Alarms** to monitor repeated failures.

With this, you've set up a fully automated ENI failover system.