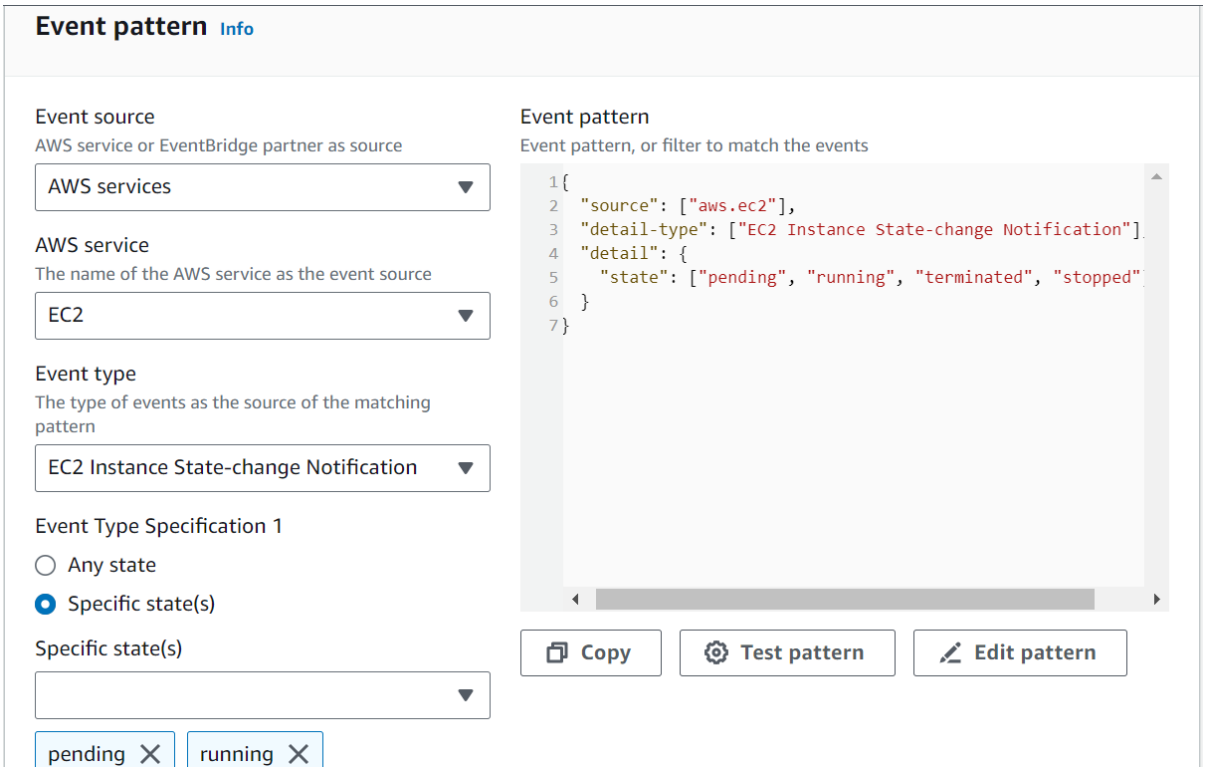
AWS – Assignments 2

Create a CloudWatch event rule to monitor ec2 instances – pending, running, stopped, terminated.

- In the AWS Management Console, type "CloudWatch" in the search bar.
    - Create a CloudWatch Event Rule
    - On the Rules page, click on the Create rule button at the top right corner of the page.
    - Select Event Source as AWS events.
    - From the Service Name dropdown, select EC2.
    - In the Event Type dropdown, select EC2 Instance State-change Notification.
    - Click on the Edit button under Event Source.
    - Select Event Pattern and choose Custom pattern (JSON editor).
    - Enter the filter for specific state changes:
    - Click Save after entering the JSON pattern.
    - In the Target section, click on the Add target button.
    - Select Lambda function from the Target dropdown.
    - Choose an existing Lambda function.
    - Click on Create a new role for this specific resource under the Existing role section.
    - This will automatically create the necessary IAM role with the required permissions for the rule to invoke the Lambda function.
    - Enter a name for your rule; Optionally, add a description for better understanding.
    - Ensure the State is set to Enabled.
    - Review and Create:



- **Part 2: Create a Lambda Function**
- In the AWS Management Console, type "Lambda" in the search bar.
- Select Lambda from the results to open the Lambda dashboard.
- Click on the Create function button.
- Choose Author from scratch.
- Function name: myec2instancefunction
- Runtime: Choose the appropriate runtime (e.g., Python 3.12).
- Role: Choose Create a new role with basic Lambda permissions.
- Click Create function.
- In the Lambda function configuration page, go to the Function code section.

```
import json
import boto3
from datetime import datetime


dynamodb = boto3.resource('dynamodb')
sns = boto3.client('sns')
```

```
table_name = 'ec2-instance'
sns_topic_arn = 'arn:aws:sns:us-east-1:654654544380:ec2-sns-topic'

def lambda_handler(event, context):
    table = dynamodb.Table(table_name)

    instance_id = event['detail']['instance-id']
    instance_state = event['detail']['state']
    instance_launch_time = event['time']

    instance_launch_time_formatted = datetime.strptime(instance_launch_time, '%Y-
%m-%dT%H:%M:%SZ').strftime('%Y-%m-%d %H:%M:%S')

    response = table.get_item(Key={'instance_id': instance_id})

    if 'Item' not in response:
        table.put_item(
            Item={
                'instance_id': instance_id,
                'instance_launch_time': instance_launch_time_formatted,
                'instance_current_state': instance_state
            }
        )
        sns.publish(
            TopicArn=sns_topic_arn,
            Message=f'New EC2 instance launched: {instance_id}, State:
{instance_state}',
            Subject='EC2 Instance Launched'
        )
    else:
        table.update_item(
            Key={'instance_id': instance_id},
            UpdateExpression='SET instance_current_state = :val1',
            ExpressionAttributeValues={':val1': instance_state}
        )

    return {
        'statusCode': 200,
        'body': json.dumps('Success')
    }
```

Environment Variables:

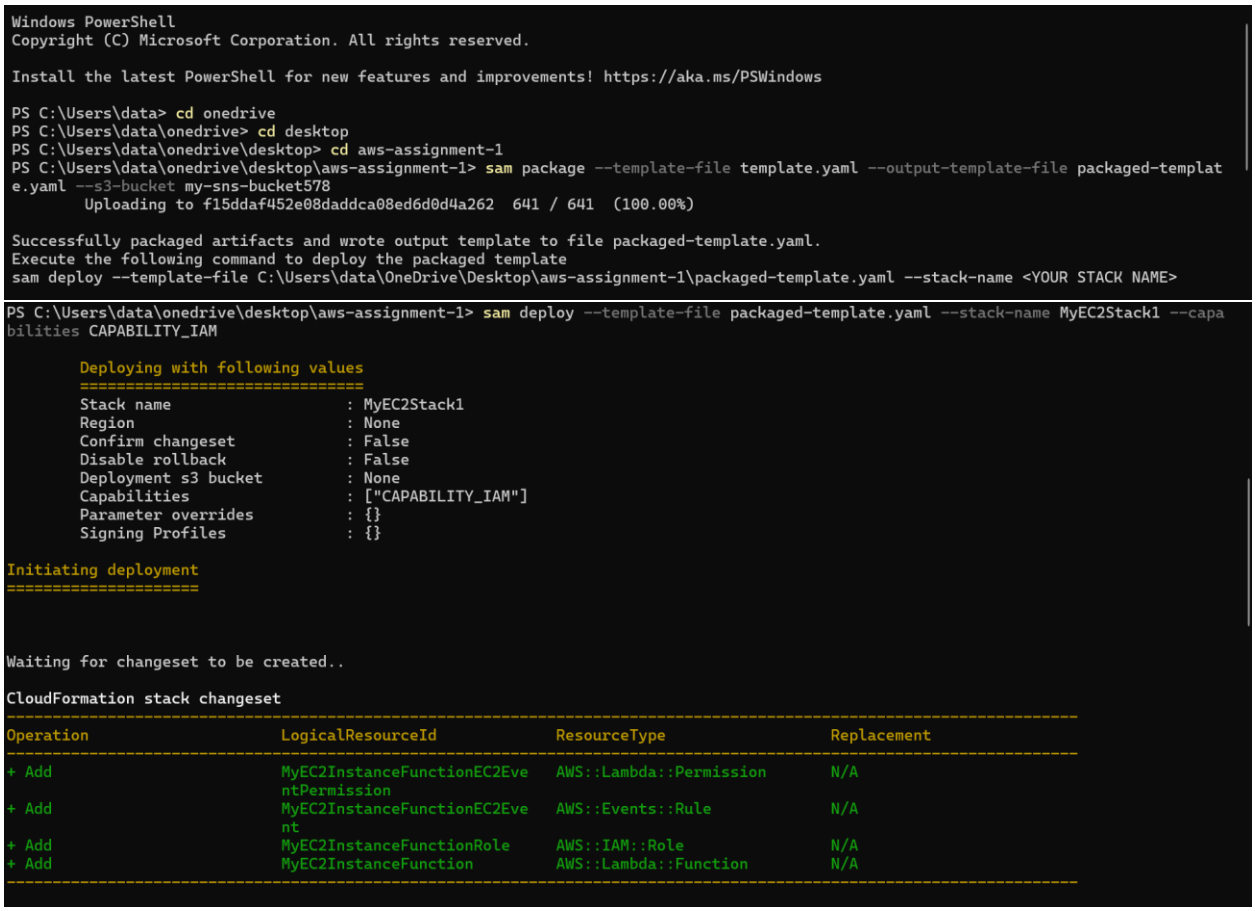
- Configure environment variables for TABLE_NAME and SNS_TOPIC_ARN
- Click the Deploy button to save your changes.
- Add Permissions for DynamoDB and SNS
- Go to the Permissions tab of your Lambda function.
- Click on the Execution role link.
- Attach the following policies to the role:
- AmazonDynamoDBFullAccess (or a custom policy with necessary DynamoDB permissions).
- AmazonSNSFullAccess (or a custom policy with necessary SNS permissions).

- **Part 3: Create a DynamoDB Table**
- In the AWS Management Console, type "DynamoDB" in the search bar.
- Select DynamoDB from the results to open the DynamoDB dashboard.
- Click on the Create table button.
- Table Name: Enter the table name.
- Partition key: instance_id (String).
- Click Create.

- **Part 4: Create an SNS Topic**
- In the AWS Management Console, type "SNS" in the search bar.
- Select Simple Notification Service (SNS) from the results to open the SNS dashboard.
- Click on the Create topic button.
- Type: Standard.
- Name: Enter a name for your topic.
- In the SNS topic details page, click on the Create subscription button.

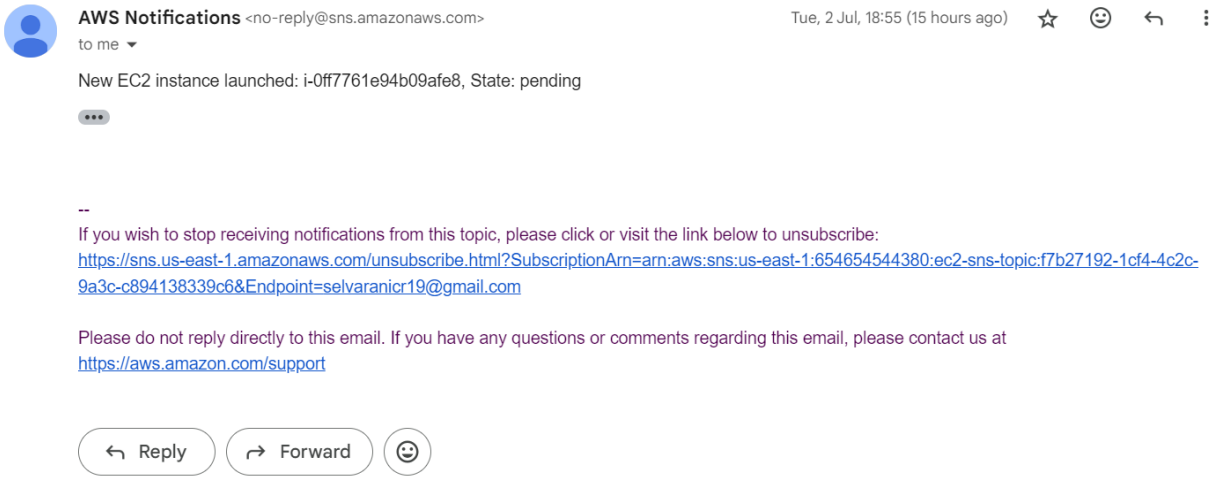
- Protocol: Choose the desired protocol (Email).
- Endpoint: Enter the appropriate endpoint (your email address).
- Click Create subscription.

- **Part 5: Deploy the Solution Using AWS SAM**



```
Changeset created successfully. arn:aws:cloudformation:us-east-1:654654544380:changeSet/samcli-deploy1719926061/cd033ce4-d5d6-4100-b791-bedad3e7670d


2024-07-02 18:44:31 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 5.0 seconds)
---------------------------------------------------------------------------------------------------------------------------
ResourceStatus                ResourceType                  LogicalResourceId             ResourceStatusReason
---------------------------------------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS            AWS::CloudFormation::Stack    MyEC2Stack1                   User Initiated
CREATE_IN_PROGRESS            AWS::IAM::Role                MyEC2InstanceFunctionRole     -
CREATE_IN_PROGRESS            AWS::IAM::Role                MyEC2InstanceFunctionRole     Resource creation Initiated
CREATE_COMPLETE               AWS::IAM::Role                MyEC2InstanceFunctionRole     -
CREATE_IN_PROGRESS            AWS::Lambda::Function         MyEC2InstanceFunction         -
CREATE_IN_PROGRESS            AWS::Lambda::Function         MyEC2InstanceFunction         Resource creation Initiated
CREATE_IN_PROGRESS            AWS::Lambda::Function         MyEC2InstanceFunction         Eventual consistency check
                                                                                          initiated
CREATE_IN_PROGRESS            AWS::Events::Rule             MyEC2InstanceFunctionEC2Eve   -
                                                            nt
CREATE_IN_PROGRESS            AWS::Events::Rule             MyEC2InstanceFunctionEC2Eve   Resource creation Initiated
                                                            nt
CREATE_COMPLETE               AWS::Lambda::Function         MyEC2InstanceFunction         -
CREATE_COMPLETE               AWS::Events::Rule             MyEC2InstanceFunctionEC2Eve   -
                                                            nt
CREATE_IN_PROGRESS            AWS::Lambda::Permission       MyEC2InstanceFunctionEC2Eve   -
                                                            ntPermission
CREATE_IN_PROGRESS            AWS::Lambda::Permission       MyEC2InstanceFunctionEC2Eve   Resource creation Initiated
                                                            ntPermission
CREATE_COMPLETE               AWS::Lambda::Permission       MyEC2InstanceFunctionEC2Eve   -
                                                            ntPermission
CREATE_COMPLETE               AWS::CloudFormation::Stack    MyEC2Stack1                   -
---------------------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------------------

CloudFormation outputs from deployed stack
---------------------------------------------------------------------------------------------------------------------------
Outputs
---------------------------------------------------------------------------------------------------------------------------
Key                 MyEC2InstanceFunctionArn
Description         ARN of the MyEC2InstanceFunction Lambda function
Value               arn:aws:lambda:us-east-1:654654544380:function:MyEC2Stack1-MyEC2InstanceFunction-EVUzGnTNx1fb

Key                 MyEC2InstanceFunctionName
Description         Name of the MyEC2InstanceFunction Lambda function
Value               MyEC2Stack1-MyEC2InstanceFunction-EVUzGnTNx1fb
---------------------------------------------------------------------------------------------------------------------------


Successfully created/updated stack - MyEC2Stack1 in None
```

- when ec2 is launched, the sns sends a notification to the given mail id.



AWS Notifications <no-reply@sns.amazonaws.com> Tue, 2 Jul, 18:55 (15 hours ago)

to me

New EC2 instance launched: i-0ff7761e94b09afe8, State: pending

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:654654544380:ec2-sns-topic:f7b27192-1cf4-4c2c-9a3c-c894138339c6&Endpoint=selvaranicr19@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at
https://aws.amazon.com/support

Reply Forward

Create a CloudFormation template in YAML format with the following components:

- Parameters: Accepts the VPC ID as input.
- IAM Role: Allows the Lambda function to execute and access required resources.
- Lambda Function: Custom resource logic to check VPC existence and return its CIDR block.
- Custom Resource: Invokes the Lambda function.
- Security Group: Created within the specified VPC with an ingress rule for SSH access.

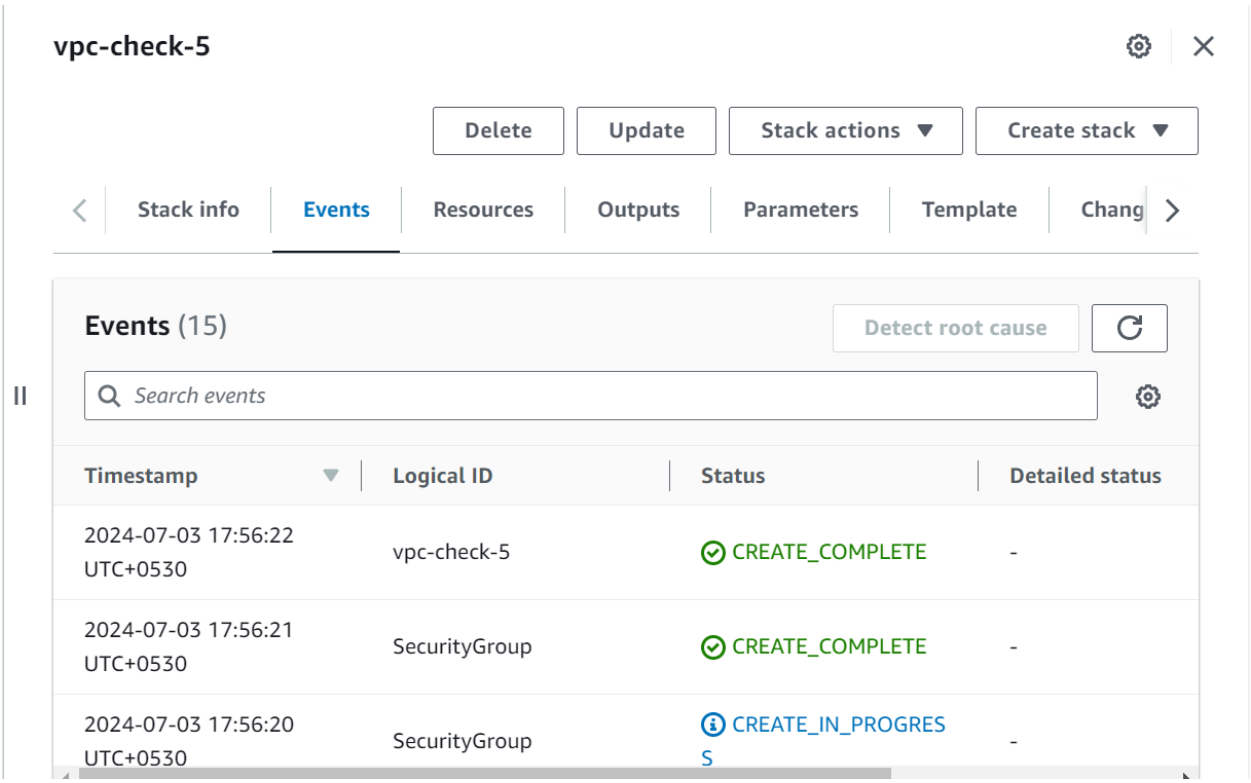
```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Task - check VPC and Create SG using Custom Resource helper

Parameters:
  VpcId:
    Type: String
    Description: ID of the VPC to check.

Resources:
  CustomResourceFunctionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      Policies:
        - PolicyName: LambdaExecutionPolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - ec2:DescribeVpcs
                Resource: '*'
```

```
        - PolicyName: CloudWatchLogsPolicy
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action:
                  - "logs:CreateLogGroup"
                  - "logs:CreateLogStream"
                  - "logs:PutLogEvents"
                Resource: "arn:aws:logs:*:*:*"

  CustomResourceFunction:
    Type: AWS::Lambda::Function
    Properties:
      Handler: index.handler
      Runtime: python3.12
      Role: !GetAtt CustomResourceFunctionRole.Arn
      Timeout: 60
      Code:
        ZipFile: |
          import cfnresponse
          import boto3
          def handler(event, context):
              try:
                  vpc_id = event['ResourceProperties']['VpcId']
                  ec2 = boto3.client('ec2')
                  response = ec2.describe_vpcs(VpcIds=[vpc_id])
                  cidr_ip = response['Vpcs'][0]['CidrBlock']
                  cfnresponse.send(event, context, cfnresponse.SUCCESS,
{"CidrIp": cidr_ip})
              except Exception as e:
                  print("Error: ", e)
                  cfnresponse.send(event, context, cfnresponse.FAILED, {"no vpc
found in your accounnt": str(e)})

  CustomResource:
    Type: Custom::VpcExists
    Properties:
      ServiceToken: !GetAtt CustomResourceFunction.Arn
      VpcId: !Ref VpcId

  SecurityGroup:
```

```
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Security group for SSH access
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: !GetAtt CustomResource.CidrIp

Outputs:
  VpcCidrBlock:
    Description: CIDR IP block of the specified VPC
    Value: !GetAtt CustomResource.CidrIp
```

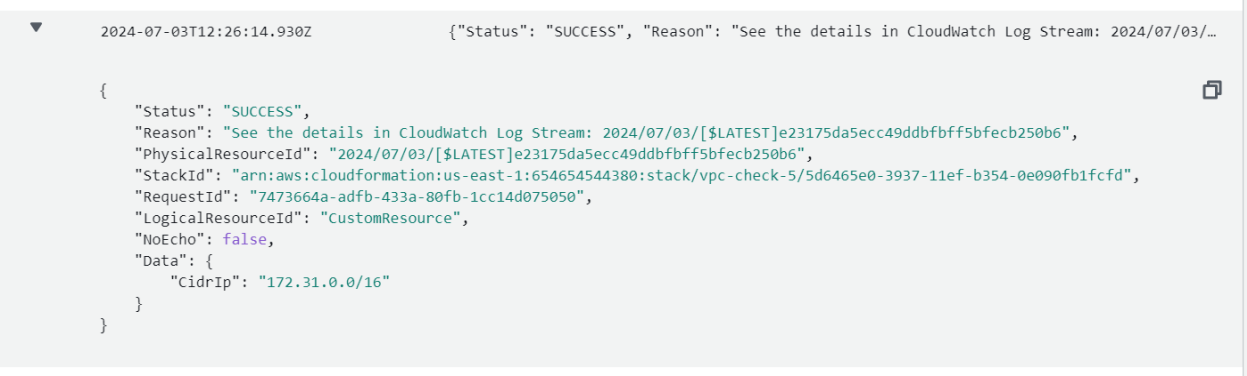
Navigate to the CloudFormation service.

- Click on "Create stack" and choose "With new resources (standard)".
- Upload the CloudFormation template file.
- Provide a stack name and enter the VPC ID in the VpcId parameter field.
- Click "Next" and configure stack options as needed.
- Review the stack configuration and click "Create stack".



Verify Lambda Function Execution

- Find the Lambda function created by the CloudFormation stack.
- Check the function's CloudWatch Logs to verify execution:
- Log in to the CloudWatch service.
- Navigate to Logs.
- Locate the log group for the Lambda function.
- Inspect the latest log stream for details.
- Confirm Custom Resource Response
- Navigate back to the CloudFormation stack.
- Check the stack events to confirm the Custom Resource execution:
- Look for a SUCCESS status in the event log.
- Verify that the CIDR IP block of the specified VPC is returned in the response data.



- Validate Security Group Creation
- Navigate to the EC2 service in the AWS Management Console.
- Go to the Security Groups section.
- Locate the Security Group created by the CloudFormation stack.
- Verify the ingress rule:
- Ensure port 22 is open.
- Check that the source IP range matches the CIDR block received from the Custom Resource.

- CloudFormation Outputs:
- VpcCidrBlock: Contains the CIDR IP block of the specified VPC.

Create a SAM template to create an AMI from existing instance id and update that to dynamodb using step fucntion

- Define prameters
- Define dynamodb
  - AttributeName: Defines the name of the attribute.
  - AttributeType: Defines the type of the attribute (S for string).
  - KeyType: Defines whether the attribute is a partition key (HASH) or sort key (RANGE).
  - BillingMode: Set to PAY_PER_REQUEST to handle billing based on the read and write requests.
  - Define IAM Roles and Policies: IAM roles for Step Functions and Lambda functions with the necessary permissions.

- Define lambda function and step function.

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Step Function for EC2 Instance Management

Parameters:
  DynamoDBTable:
    Type: String
    Default: "step-task-table"
    Description: Name of the DynamoDB table for Step Function execution records
  ExistingInstanceId:
    Type: String
    Default: "i-05842583641f404fc"
    Description: ID of the existing EC2 instance
  Region:
    Type: String
    Default: "us-east-1"
    Description: AWS region where the resources will be deployed
  AccountId:
    Type: String
    Description: AWS account ID

Resources:
  StepFunctionExecutionTable:
    Type: AWS::DynamoDB::Table
    Properties:
      TableName: !Ref DynamoDBTable
      AttributeDefinitions:
        - AttributeName: step_function_name
          AttributeType: S
        - AttributeName: step_function_launch_time
          AttributeType: S
      KeySchema:
        - AttributeName: step_function_name
          KeyType: HASH
        - AttributeName: step_function_launch_time
          KeyType: RANGE
      BillingMode: PAY_PER_REQUEST

  StepFunctionRole:
    Type: AWS::IAM::Role
```

```
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: states.amazonaws.com
            Action: sts:AssumeRole
      Policies:
        - PolicyName: StepFunctionPolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action: dynamodb:*
                Resource: !GetAtt StepFunctionExecutionTable.Arn
              - Effect: Allow
                Action: Lambda:*
                Resource: "*"

  CreateAmiFunctionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      Policies:
        - PolicyName: CreateAmiFunctionPolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - ec2:CreateImage
                Resource: "*"

  CreateAmiFunction:
    Type: AWS::Serverless::Function
```

```yaml
  Properties:
    Handler: create_ami_function.handler
    Runtime: python3.12
    CodeUri: .
    Timeout: 60
    Environment:
      Variables:
        Region: !Ref Region
    Role: !GetAtt CreateAmiFunctionRole.Arn

LaunchInstanceFunctionRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
          Action: sts:AssumeRole
    Policies:
      - PolicyName: LaunchInstanceFunctionPolicy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Action:
                - ec2:RunInstances
              Resource: "*"

LaunchInstanceFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: launch_instance_function.handler
    Runtime: python3.12
    CodeUri: .
    Timeout: 60
    Environment:
      Variables:
        Region: !Ref Region
    Role: !GetAtt LaunchInstanceFunctionRole.Arn
```

```
StepFunction:
  Type: AWS::StepFunctions::StateMachine
  Properties:
    Definition:
      StartAt: Step1
      States:
        Step1:
          Type: Task
          Resource: arn:aws:states:::dynamodb:putItem
          Parameters:
            TableName: !Ref DynamoDBTable
            Item:
              "step_function_name": {"S": "sample-step-function"}
              "step_function_launch_time": {"S": "${AWS::Lambda::DateTime}"}
              "existing-instance-id": {"S": !Ref ExistingInstanceId}
              "new-instance-id": {"S": ""}
              "step_function_status": {"S": "STARTED"}
          Next: Step2
        Step2:
          Type: Task
          Resource: !GetAtt CreateAmiFunction.Arn
          Parameters:
            Region: !Ref Region
            ExistingInstanceId: !Ref ExistingInstanceId
          ResultPath: "$.Step2Output"
          Catch:
            - ErrorEquals: ["States.ALL"]
              Next: HandleFailure
          Next: WaitState
        WaitState:
          Type: Wait
          Seconds: 200
          Next: Step3
        Step3:
          Type: Task
          Resource: !GetAtt LaunchInstanceFunction.Arn
          InputPath: "$"
          Parameters:
            Region: !Ref Region
            ami_id.$: "$.Step2Output.ami_id"
          Catch:
            - ErrorEquals: ["States.ALL"]
              Next: HandleFailure
```

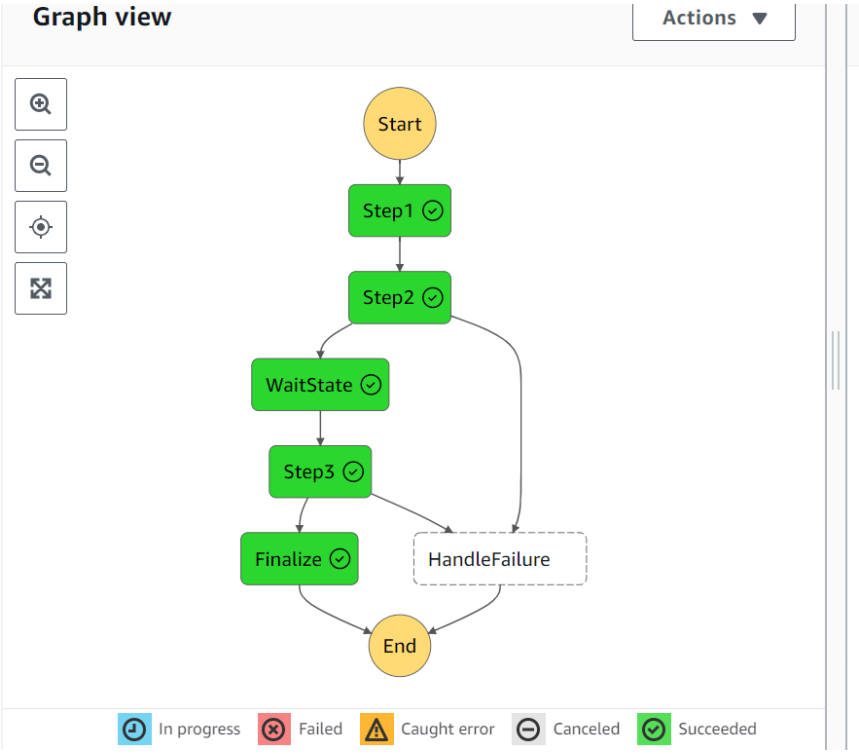
```
            Next: Finalize
          HandleFailure:
            Type: Task
            Resource: arn:aws:states:::dynamodb:updateItem
            Parameters:
              TableName: !Ref DynamoDBTable
              Key:
                step_function_name: {"S": "sample-step-function"}
                step_function_launch_time: {"S": "${AWS::Lambda::DateTime}"}
              ExpressionAttributeValues:
                ":status": {"S": "FAILED"}
              UpdateExpression: "SET step_function_status = :status"
            End: true
          Finalize:
            Type: Task
            Resource: arn:aws:states:::dynamodb:updateItem
            Parameters:
              TableName: !Ref DynamoDBTable
              Key:
                step_function_name: {"S": "sample-step-function"}
                step_function_launch_time: {"S": "${AWS::Lambda::DateTime}"}
              ExpressionAttributeValues:
                ":status": {"S": "COMPLETED"}
              UpdateExpression: "SET step_function_status = :status"
            End: true
      RoleArn: !GetAtt StepFunctionRole.Arn
```

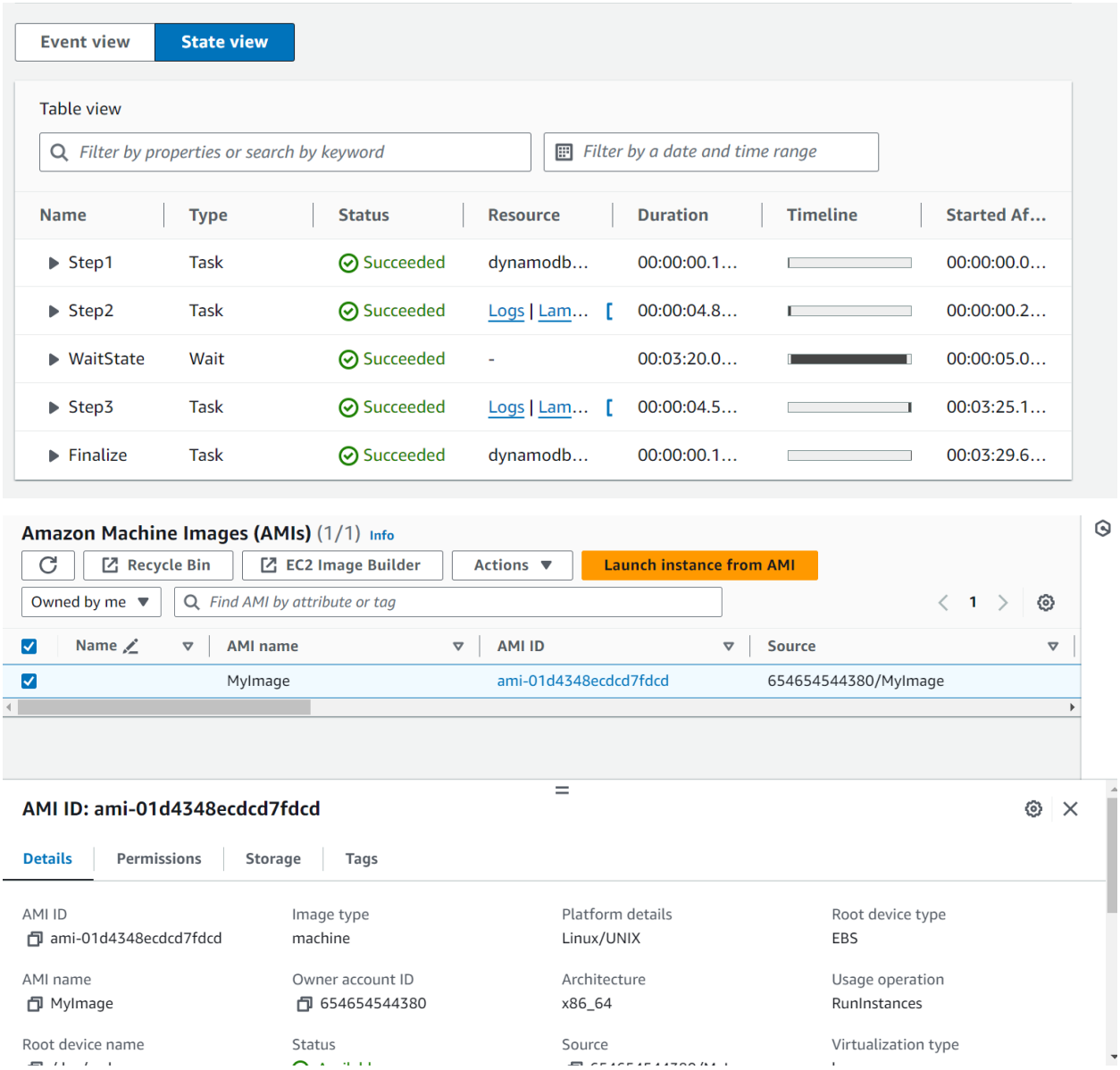
- Implemented two Lambda functions: one for creating an AMI and one for launching an EC2 instance.

```
import boto3
def handler(event, context):
    ec2 = boto3.client('ec2', region_name=event['Region'])
    instance_id = event['ExistingInstanceId']
    response = ec2.create_image(
        InstanceId=instance_id,
        Name='MyImage'
    )
    ami_id = response['ImageId']
    return {
        'ami_id': ami_id
    }
```

```python
import boto3

def handler(event, context):
    ec2 = boto3.client('ec2', region_name=event['Region'])
    response = ec2.run_instances(
        ImageId=event['ami_id'],
        MinCount=1,
        MaxCount=1,
        InstanceType='t2.micro'
    )
    instance_id = response['Instances'][0]['InstanceId']
    return {
        'instance_id': instance_id
    }
```

- Deploy the cf stack
  sam build
  same deploy –guided
- Execute the step function and verify the DynamoDB, ec2 instance and AMI.




Event view
State view
Table view
Filter by properties or search by keyword
Filter by a date and time range
Name
Type
Status
Resource
Duration
Timeline
Started Af...
Step1
Task
Succeeded
dynamodb...
00:00:00.1...
00:00:00.0...
Step2
Task
Succeeded
Logs | Lam...
00:00:04.8...
00:00:00.2...
WaitState
Wait
Succeeded
-
00:03:20.0...
00:00:05.0...
Step3
Task
Succeeded
Logs | Lam...
00:00:04.5...
00:03:25.1...
Finalize
Task
Succeeded
dynamodb...
00:00:00.1...
00:03:29.6...
Amazon Machine Images (AMIs) (1/1) Info
Recycle Bin
EC2 Image Builder
Actions
Launch instance from AMI
Owned by me
Find AMI by attribute or tag
1
Name
AMI name
AMI ID
Source
MyImage
ami-01d4348ecdcd7fdcd
654654544380/MyImage
AMI ID: ami-01d4348ecdcd7fdcd
Details
Permissions
Storage
Tags
AMI ID
ami-01d4348ecdcd7fdcd
Image type
machine
Platform details
Linux/UNIX
Root device type
EBS
AMI name
MyImage
Owner account ID
654654544380
Architecture
x86_64
Usage operation
RunInstances
Root device name
Status
Source
Virtualization type