AWS Auto Scaling Web Environment with CloudFormation

Instruction

By @Chihiro-001

Scenario

- A company wants to create a staging environment and a production environment for its website using a template. They want to be able to use the same environment for both validation and production.
- The website must be accessible via a secure connection when browsing.
- The template environment should be used as a basis for adding features such as EC2 auto-scaling and alarm notification based on status monitoring.

1 ：Requirement

Create template and setup environment with the following requirements:

- automatically build an environment that fulfills the following 1-3:
    1. Create public subnets in two Availability Zones (AZ).
    2. Deploy EC2 instances in each public subnet, using Apache as the web server.
    3. Use ALB (Application Load Balancer) and configure each EC2 instance in the target group.

The following steps are done manually after setting up the stack:

- have secure encrypted communication using a domain for the web page.
    - Hint: Issue a certificate using AWS services during step 1.
- make EC2 automatically scale out when the CPU load exceeds 70%, and scale in when it falls below 20%.
- configure so that when the threshold is exceeded in monitoring, alerts can be sent to responsible personnel via email.

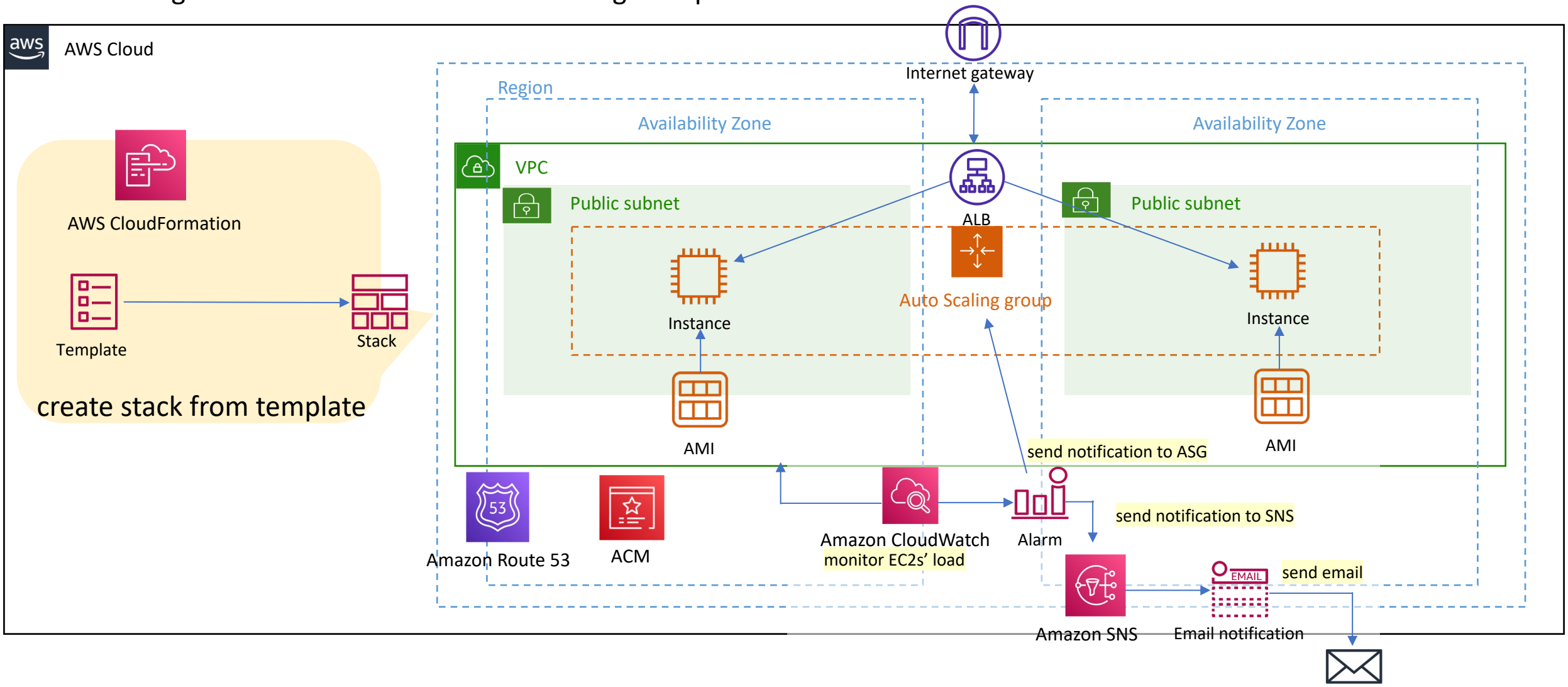
2. Design

• Specification (Solution) for Requirements:

No	Requirement	Specification (Solution)
1	Automatically build an environment that fulfills the following conditions: ① Create public subnets in two Availability Zones (AZ). ② Deploy EC2 instances in each public subnet, using Apache as the web server. ③ Use ALB to configure each EC2 instance in the target group.	Use CloudFormation to automatically build the environment. ① Create YAML files for environment construction such as VPC and Internet Gateway. ② Prepare YAML files to place AL2 AMIs that support Apache in each public subnet. ③ Use the YAML files created in step ② to configure ALB and set EC2 instances in the target group.
2	have secure encrypted communication using a domain for the web page.	As preparation, create a Route53 Hosted Zone and acquire an external domain. Configure to issue an SSL certificate in ACM for the acquired domain name during step 1.
3	make EC2 automatically scale out when the CPU load exceeds 70%, and scale in when it falls below 20%.	Use ASG (Auto Scaling Group) with Simple Scaling Policy to scale out when CPU load exceeds 70% and scale in when it falls below 20%.
4	Configure so that when the threshold is exceeded in monitoring, alerts can be sent to responsible person(s) via email.	Set up CloudWatch to monitor CPU thresholds. Set up SNS (Simple Notification Service) to send alert information via email to responsible personnel.

2. Design

Diagram: CloudFormation + Auto Scaling Group + CloudWatch + SNS

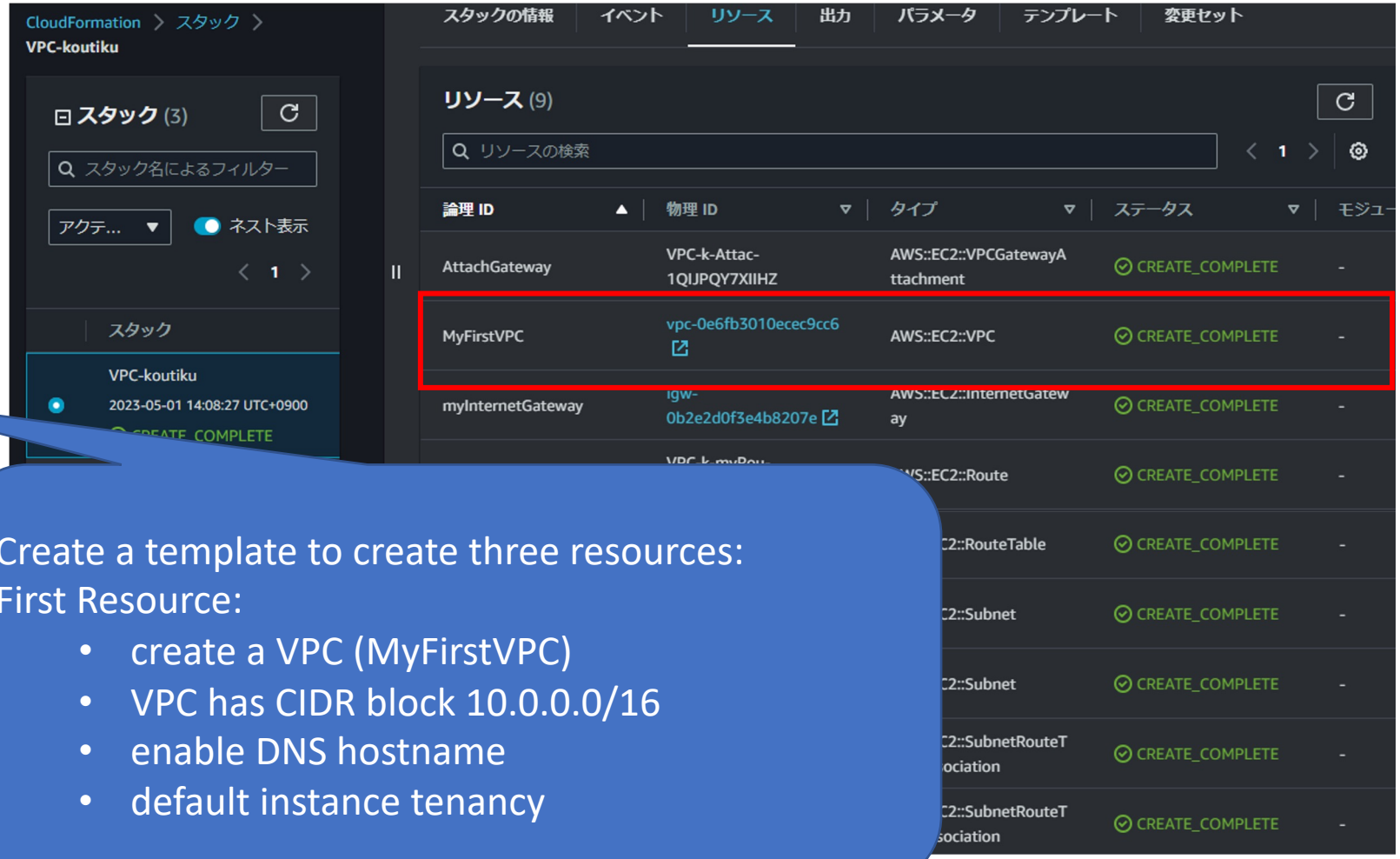


3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:
① Create public subnets in two Availability Zones (AZ). Create a template to create three resources:

Create VPC environment

```
Resources:
  MyFirstVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsSupport: 'true'
      EnableDnsHostnames: 'true'
      InstanceTenancy: default
      Tags:
      - Key: Name
        Value: CF-VPC
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref MyFirstVPC
      Tags:
      - Key: Name
        Value: CF-VPC-PublicRT

  PublicSubnet1A:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyFirstVPC
      CidrBlock: 10.0.0.0/24
```



Create a template to create three resources:
First Resource:

- create a VPC (MyFirstVPC)
- VPC has CIDR block 10.0.0.0/16
- enable DNS hostname
- default instance tenancy

3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:

① Create public subnets in two Availability Zones (AZ). Create a template to create three resources:

```
PublicSubnet1A:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyFirstVPC
    CidrBlock: 10.0.0.0/24
    MapPublicIpOnLaunch: true
    AvailabilityZone: "ap-northeast-1a"
    Tags:
    - Key: Name
      Value: CF-publicsubnet-1a
```

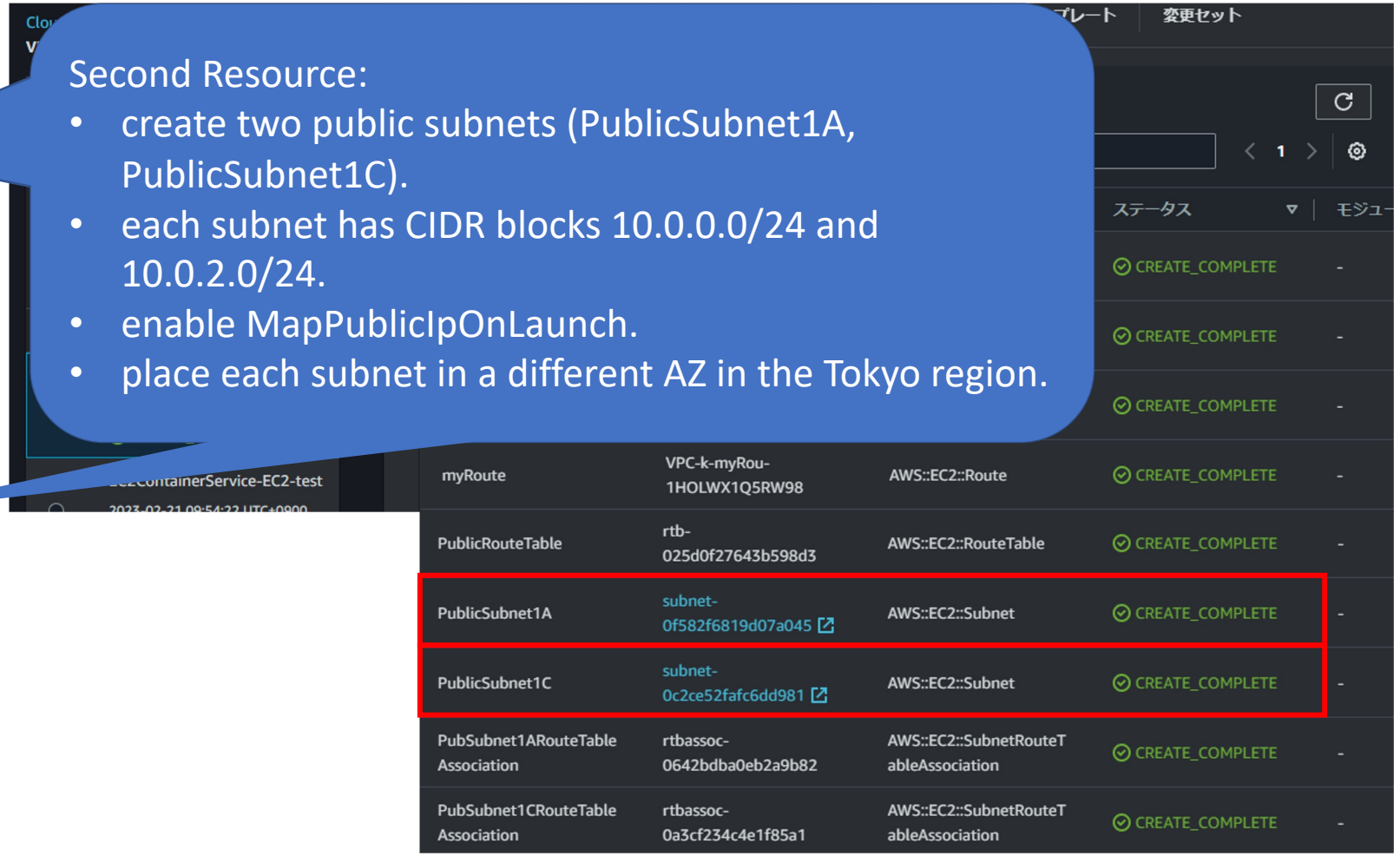
```
PubSubnet1ARouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet1A
    RouteTableId: !Ref PublicRouteTable
```

```
PublicSubnet1C:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyFirstVPC
    CidrBlock: 10.0.2.0/24
    MapPublicIpOnLaunch: true
    AvailabilityZone: "ap-northeast-1c"
    Tags:
    - Key: Name
      Value: CF-publicsubnet-1c
```

```
PubSubnet1CRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet1C
    RouteTableId: !Ref PublicRouteTable
```

Second Resource:

- create two public subnets (PublicSubnet1A, PublicSubnet1C).
- each subnet has CIDR blocks 10.0.0.0/24 and 10.0.2.0/24.
- enable MapPublicIpOnLaunch.
- place each subnet in a different AZ in the Tokyo region.

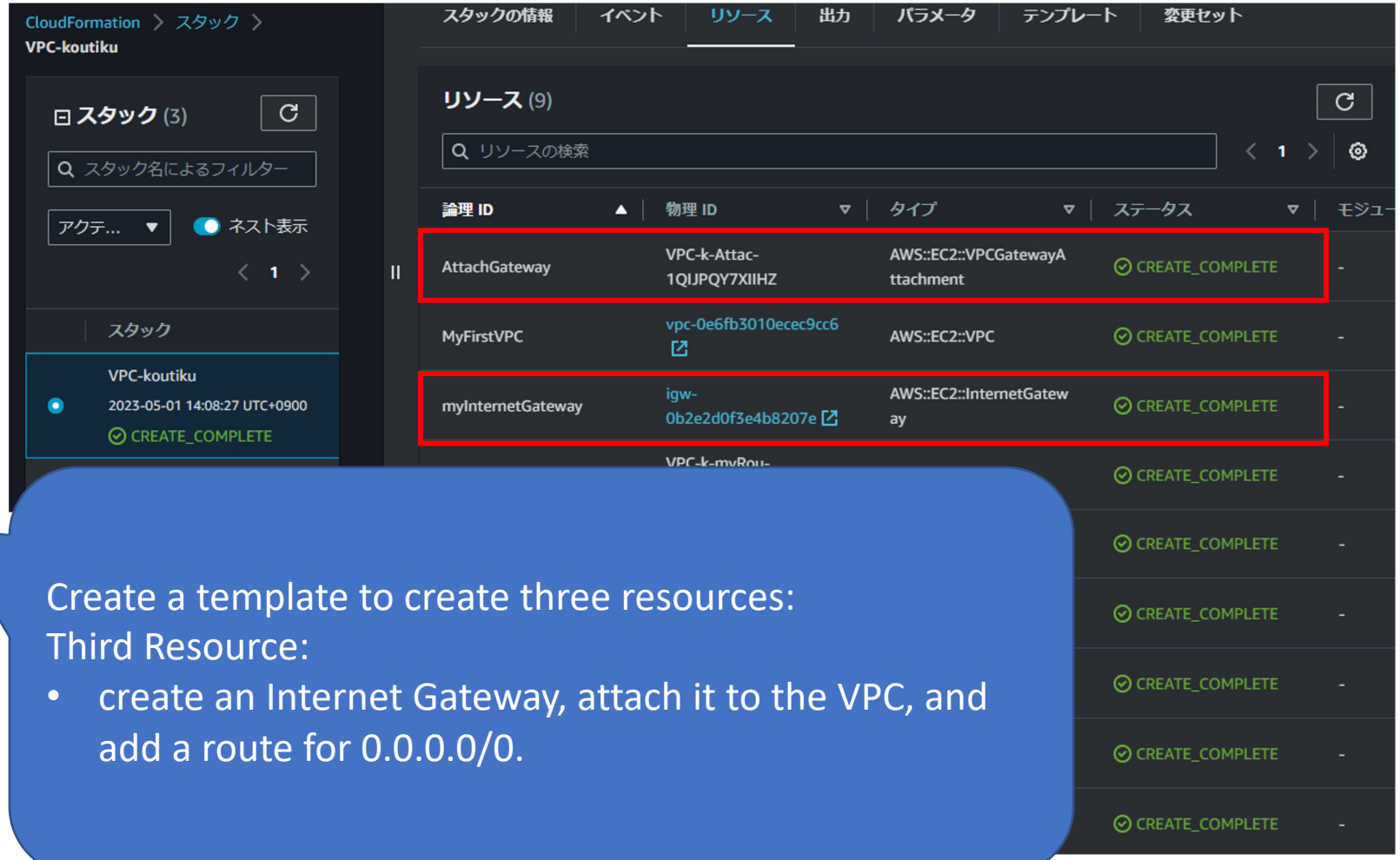


3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:
① Create public subnets in two Availability Zones (AZ). Create a template to create three resources:

Create VPC environment

```
myInternetGateway:
  Type: "AWS::EC2::InternetGateway"
  Properties:
    Tags:
    - Key: Name
      Value: CF-INGW
AttachGateway:
  Type: AWS::EC2::VPCGatewayAttachment
  Properties:
    VpcId: !Ref MyFirstVPC
    InternetGatewayId: !Ref myInternetGateway
myRoute:
  Type: AWS::EC2::Route
  DependsOn: myInternetGateway
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref myInternetGateway
```



Create a template to create three resources:
Third Resource:

- create an Internet Gateway, attach it to the VPC, and add a route for 0.0.0.0/0.

3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:
① Create public subnets in two Availability Zones (AZ). Create a template to create three resources:

Create VPC environment

```
Outputs:
  StackVPC:
    Description: The ID of the VPC
    Value: !Ref MyFirstVPC
    Export:
      Name: !Sub "${AWS::StackName}-VPCID"
  StackPublicSubnet1A:
    Description: The ID of the VPC Subnet
    Value: !Ref PublicSubnet1A
    Export:
      Name: !Sub "${AWS::StackName}-PublicSubnet1A"
  StackPublicSubnet1C:
    Description: The ID of the VPC Subnet
    Value: !Ref PublicSubnet1C
    Export:
      Name: !Sub "${AWS::StackName}-PublicSubnet1C"
```

1

1. export the ID of the created VPC.

2

2. export the ID of the created PublicSubnet1A.

3

3. export the ID of the created PublicSubnet1C.

Exported IDs can be used in other CloudFormation stacks

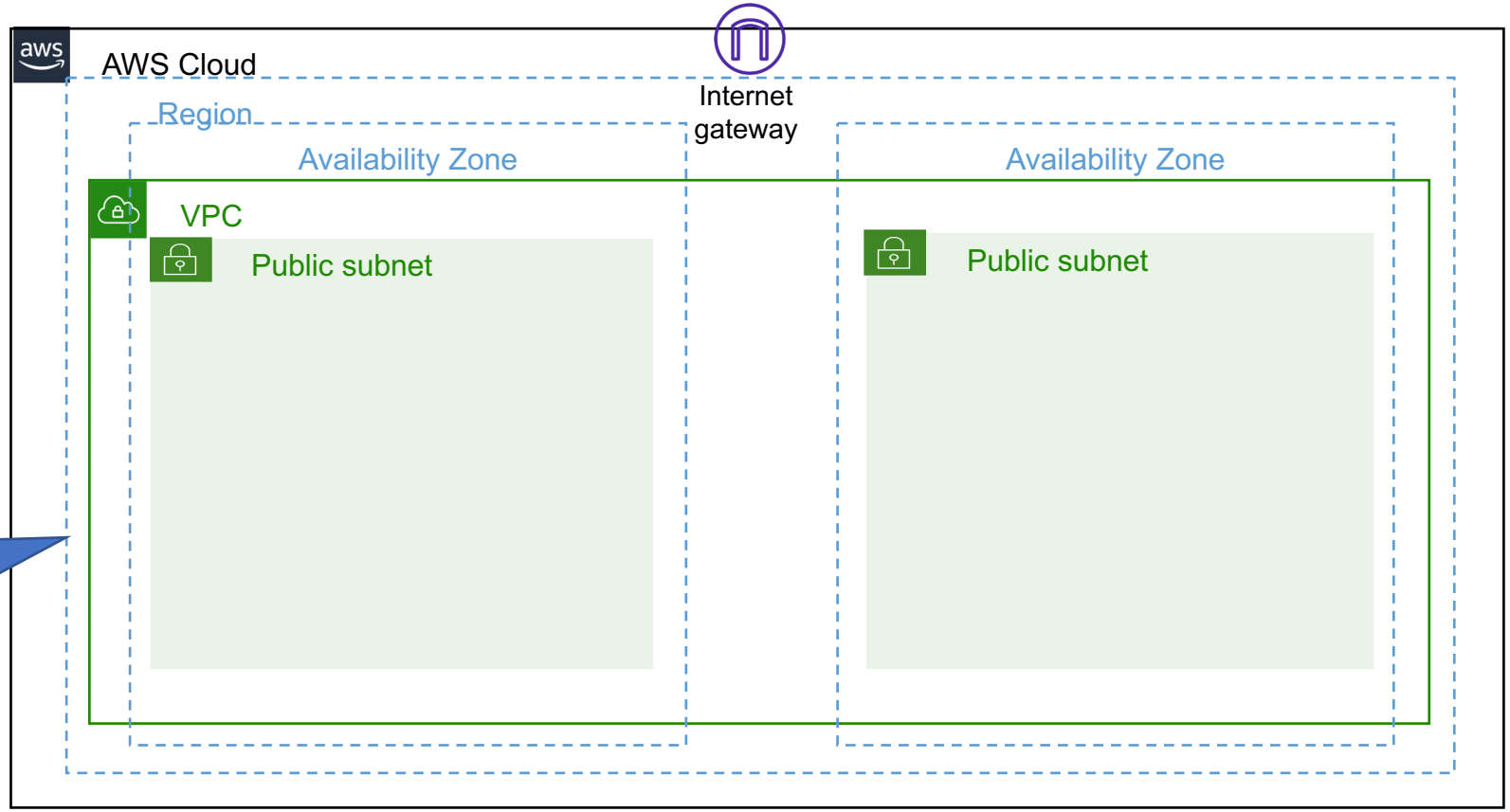
3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:

① Create public subnets in two Availability Zones (AZ). Create a template to create three resources:

Create VPC environment

Now it looks like this



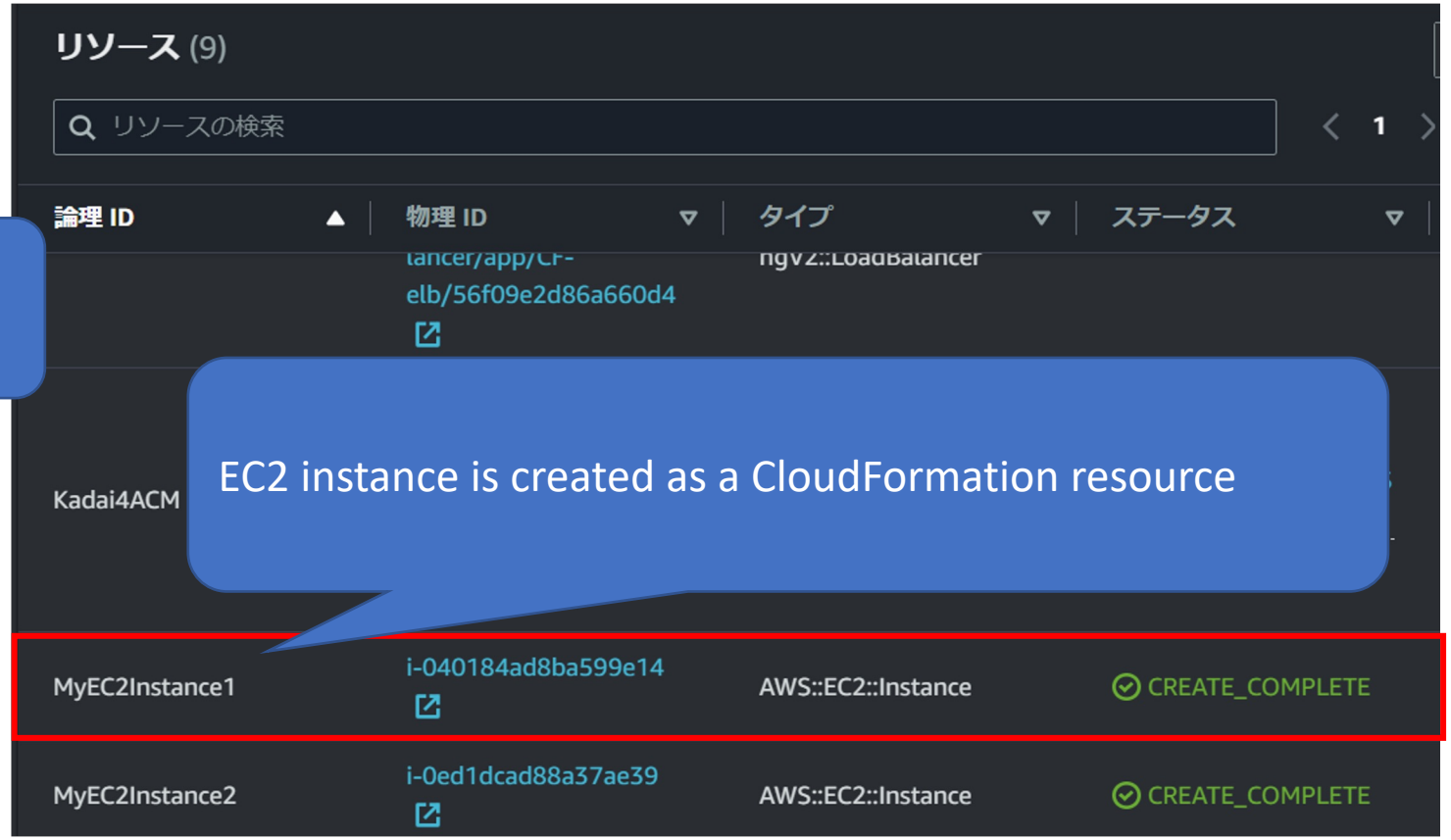
3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:
②-a: Deploy EC2 instances in each public subnet.

```
MyEC2Instance1:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: !Ref EC2AMI
    InstanceType: t2.micro
    SubnetId: !ImportValue VPC-koutiku-PublicSubnet1A
    UserData: !Base64
      #!/bin/bas
      systemctl start httpd.service
    BlockDeviceMappings:
      - DeviceName: /dev/xvda
        Ebs:
          VolumeType: gp2
          VolumeSize: 8
    Tags:
      - Key: Name
        Value: CF-public-1a
    KeyName: !Ref KeyName
    SecurityGroupIds:
      - !GetAtt "InstanceSecurityGroup.GroupId"
```

Refer to the SubnetId from the stack where the VPC was built.
Note: Output the SubnetId beforehand.

Create a separate stack to create EC2 instances.

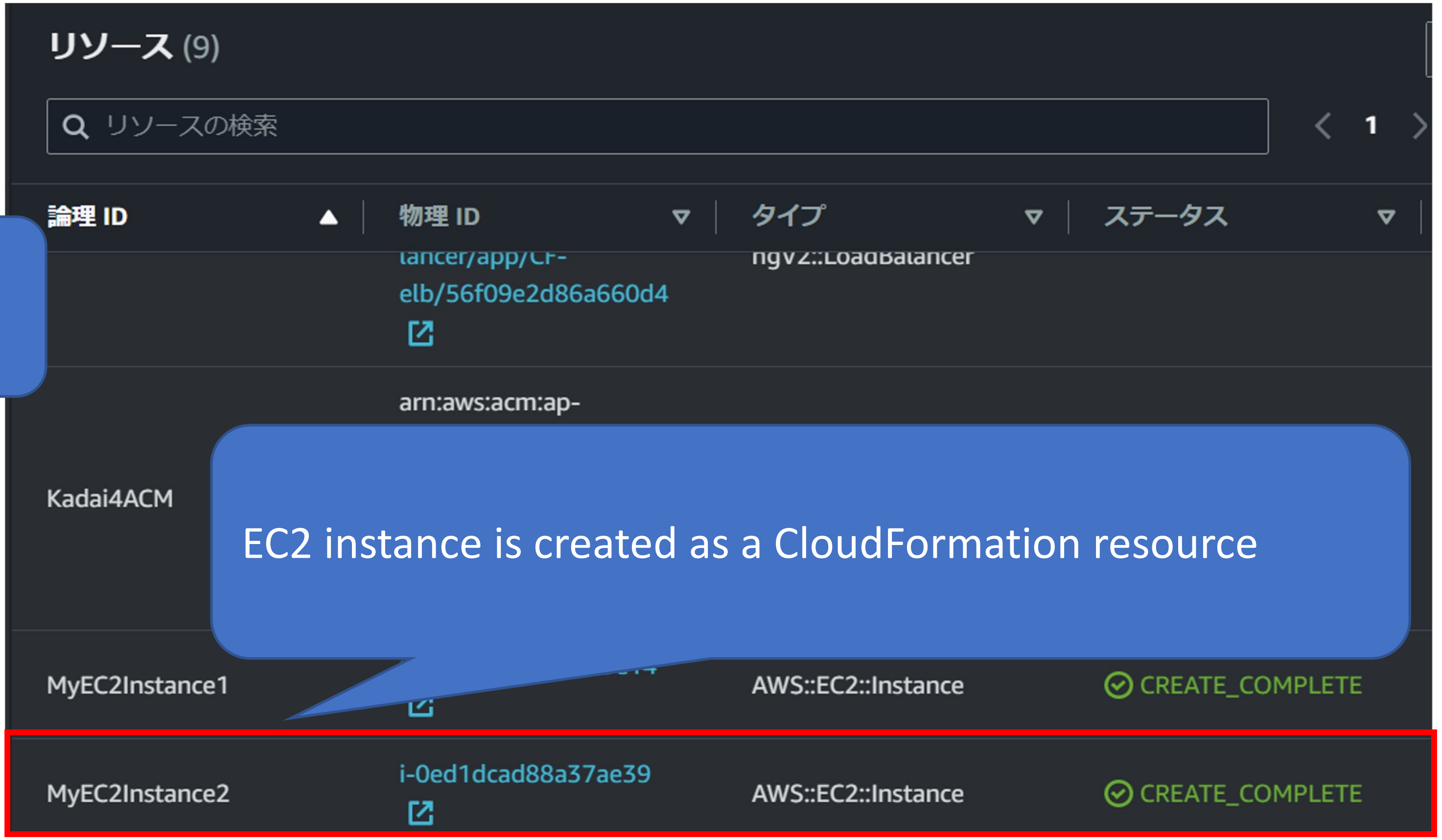


3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:
②-a: Deploy EC2 instances in each public subnet.

```
MyEC2Instance2:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: !Ref EC2AMI
    InstanceType: t2.micro
    SubnetId: !ImportValue VPC-koutiku-PublicSubnet1C
    UserData:
      ...
      systemctl start httpd.service
    BlockDeviceMappings:
      - DeviceName: /dev/xvda
        Ebs:
          VolumeType: gp2
          VolumeSize: 8
    Tags:
      - Key: Name
        Value: CF-public-1c
    KeyName: !Ref KeyName
    SecurityGroupIds:
      - !GetAtt "InstanceSecurityGroup.GroupId"
```

Refer to the SubnetId from the stack where the VPC was built.
Note: Output the SubnetId beforehand.



EC2 instance is created as a CloudFormation resource

3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:
②-b: Use Apache as the web server

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Create EC2 Instance
Parameters:
  KeyName:
    Description: Name of an existing EC2 KeyPair.
    Type: AWS::EC2::KeyPair::KeyName
    ConstraintDescription: Can contain only ASCII characters.
  EC2AMI:
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2
```

To ensure Apache is available, use the Amazon Linux 2 AMI. Utilize AWS Systems Manager Parameter Store to fetch the latest Amazon Linux AMI.

```
MyEC2Instance1:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: !Ref EC2AMI
    InstanceType: t2.micro
    SubnetId: !ImportValue VPC-koutiku
    UserData: !Base64 |
      #!/bin/bash
      yum update -y
      yum -y install httpd
      touch /var/www/html
      chown apache:apache
      systemctl enable ht
      systemctl start htt
    BlockDeviceMappings:
      - DeviceName: /dev/
        Ebs:
          VolumeType: gp2
          VolumeSize: 8
    Tags:
      - Key: Name
        Value: CF-public-
    KeyName: !Ref KeyName
    SecurityGroupIds:
      - !GetAtt "Instance
```

Install Apache using the Yum command:
yum -y install httpd

```
MyEC2Instance2:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: !Ref EC2AMI
    InstanceType: t2.micro
    SubnetId: !ImportValue VPC-koutiku-PublicSubnet1C
    UserData: !Base64 |
      #!/bin/bash
      yum update -y
      yum -y install httpd
      touch /var/www/html/.check_alive
      chown apache:apache -R /var/www/html
      systemctl enable httpd.service
      systemctl start httpd.service
```

3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:
③: Use ALB and configure each EC2 instance in the target group.

③ Configure the ALB

```
InternetELB:
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Properties:
    Name: CF-elb
    SecurityGroups:
      - !Ref InstanceSecurityGroup
    Subnets:
      - !ImportValue VPC-koutiku-PublicSubnet1A
      - !ImportValue VPC-koutiku-PublicSubnet1C
```

InternetELB
- configure the ALB.
- set the ALB's security group to the one specified for EC2.
- import the Subnets Output from the stack where the VPC was built.

```
ELBtargetGP:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    Name: ELBtargetGP
    Port: 80
    Protocol: HTTP
    Targets:
      - Id: !Ref MyEC2Instance1
      - Id: !Ref MyEC2Instance2
    TargetType: instance
    VpcId: !ImportValue VPC-koutiku-VPCID
```

ELBtargetGP
- configure the ALB's target group.
- specify Port 80 with the HTTP protocol.
- specify the EC2 instances to which the ELB forwards requests.

3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:
③: Use ALB and configure each EC2 instance in the target group.

③ Configure the ALB

```yaml
ELBListenerHTTP: # 80=>443に転送
  Type: AWS::ElasticLoadBalancingV2::Listener
  DependsOn:
    - InternetELB
  Properties:
    DefaultActions:
      - Type: redirect
        RedirectConfig:
          Protocol: HTTPS
          Port: 443
          Host: '#{host}'
          Path: '/#{path}'
          Query: '#{query}'
          StatusCode: HTTP_301
    LoadBalancerArn: !Ref InternetELB
    Port: 80
    Protocol: HTTP
```

1

```yaml
ELBListenerHTTPS: # SSL設定してTargetGroupに転送
  Type: AWS::ElasticLoadBalancingV2::Listener
  DependsOn: InternetELB
  Properties:
    Certificates:
      - CertificateArn: !Ref Kadai4ACM
    DefaultActions:
      - TargetGroupArn: !Ref ELBtargetGP
        Type: forward
    LoadBalancerArn: !Ref InternetELB
    Port: 443
    Protocol: HTTPS
    SslPolicy: ELBSecurityPolicy-2016-08
```

2

```yaml
ListenerRule1:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    ListenerArn: !Ref ELBListenerHTTPS
    Priority: 2
    Conditions:
      - Field: host-header
        HostHeaderConfig:
          Values:
            - !Ref DomainName
    Actions:
      - Type: forward
        TargetGroupArn: !Ref ELBtargetGP
```

3

1

ELBListenerHTTP:

- Create a listener for HTTP port 80.
- Set up HTTPS redirect.

2

ELBListenerHTTPS:

- Create a listener for HTTPS port 443.
- Specify an SSL certificate and configure forwarding to the target group.

3

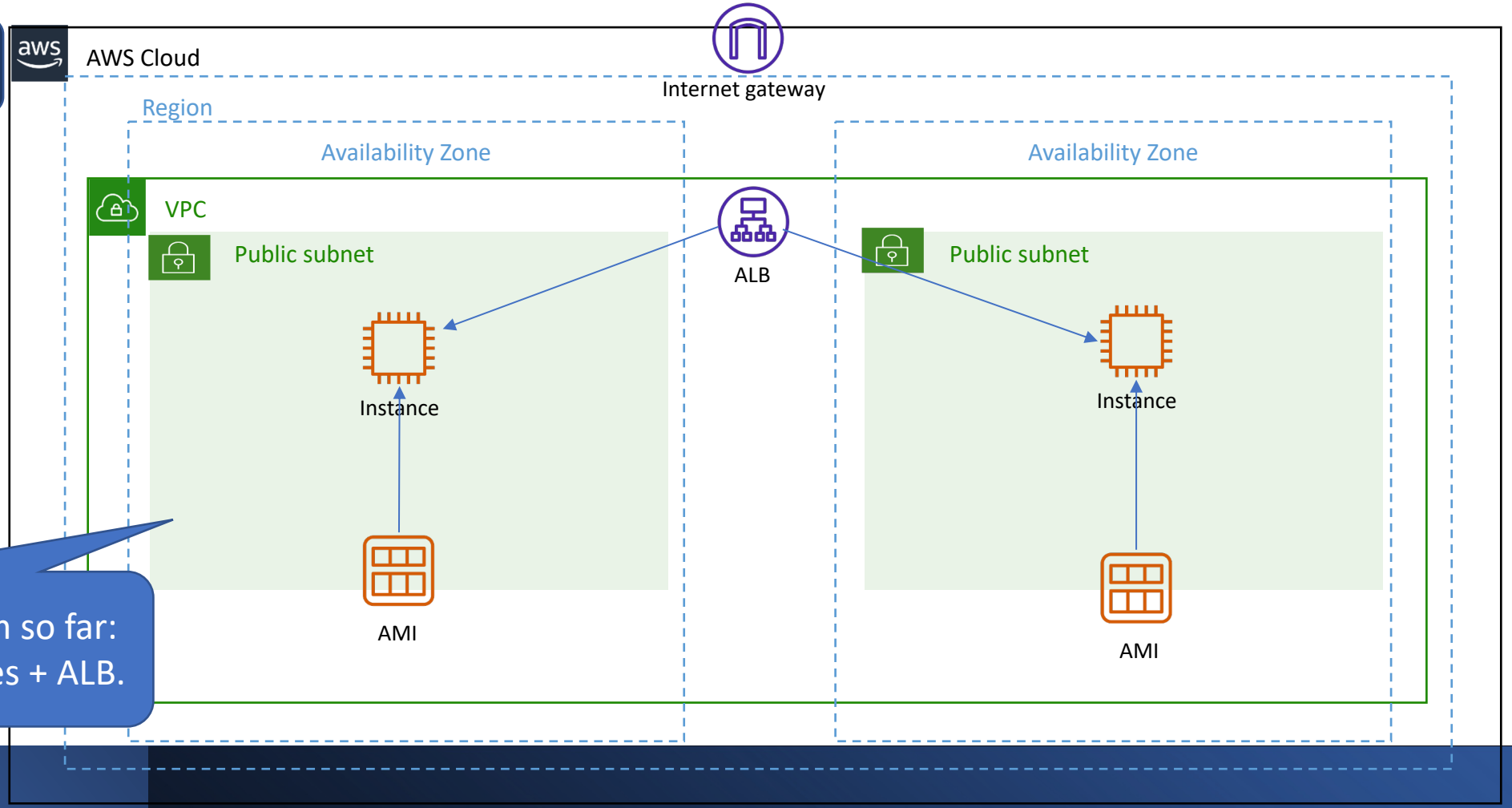
ListenerRule1:

- Create a listener rule.
- Configure forwarding to the target group if the Host header of the HTTP request matches the DomainName.

3. Implementation

Requirement 1: Automatically build an environment that fulfills the following conditions:
③: Use ALB and configure each EC2 instance in the target group.

Set up EC2 and ALB in the VPC environment.



Here's the configuration diagram so far:
VPC Environment + EC2 Instances + ALB.

3. Implementation

Requirement 2: Enable secure encrypted communication using a domain for the web page.

Preparation Step ①: Setting up Route53

Steps
- create a hosted zone in Route53.
- purchase a domain name from "お名前.com" (Onamae.com)
- issue a certificate in ACM (AWS Certificate Manager)

During the preparation phase, there will be two records:
- NS record
- SOA record

3. Implementation

Requirement 2: Enable secure encrypted communication using a domain for the web page.

preparation Step ②: Purchase a domain name from "お名前.com" (Onamae.com)

Route53

Register the value of the NS record for "yumetech.net" as the nameserver for "お名前.com" (Onamae.com).

3. Implementation

Requirement 2: Enable secure encrypted communication using a domain for the web page.

Preparation Step ③:
Issue a certificate in ACM


```
Parameters:
  KeyName:
    Description: Name of an existing EC2 KeyPair.
    Type: AWS::EC2::KeyPair::KeyName
    ConstraintDescription: Can contain only ASCII characters.
  EC2AMI:
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2
  HostZoneId:
    Description: FQDN of the hosted zone.
    Type: String
    Default: 'Z03917251J40DXFLPJW6J'
  DomainName:
    Description: FQDN of the HostZone yumetech.net
```

Set the Hosted Zone ID to Default.

3. Implementation

Requirement 2: Enable secure encrypted communication using a domain for the web page.

Preparation Step ③:
Issue a certificate in ACM

```
HostZoneId:
  Description: FQDN of the hosted zone.
  Type: String
  Default: 'Z03917251J40DXFLPJW6J'
DomainName:
  Description: FQDN of the HostZone yumetech.net
  Type: String
  Default: yumetech.net
SubjectAlternativeName:
  Description: FQDN of the Subdomains.
  Type: String
  Default: '*.yumetech.net'
SubDomain:
  Description: FQDN of the Subdomain
  Type: String
  Default: www.yumetech.net
```

1

1: Define an FQDN to be protected by ACM.

2

2: Define additional FQDN

3

3: Define a subdomain.

Ensure users can access www.yumetech.net

3. Implementation

Requirement 2: Enable secure encrypted communication using a domain for the web page.

```
Domain1Record: # yumetech.net
  Type: AWS::Route53::RecordSetGroup
  Properties:
    HostedZoneId: !Sub ${HostZoneId}
    Comment: Zone apex alias targeted to public LoadBalancer.
    RecordSets:
      - Name: !Sub ${DomainName}
        Type: A
        AliasTarget:
          HostedZoneId: !GetAtt InternetELB.CanonicalHostedZoneID
          DNSName: !GetAtt InternetELB.DNSName
```

1

1: Create a Type A record.

```
Domain1SubDomainRecord: # www.yumetec.net
  Type: AWS::Route53::RecordSet
  DependsOn: InternetELB
  Properties:
    HostedZoneId: !Sub '${HostZoneId}'
    Name: !Sub '${SubDomain}'
    Type: CNAME
    TTL: 60
    ResourceRecords:
      - !GetAtt InternetELB.DNSName
```

2

2: Create a CNAME record.

```
# ---------------------
# ACM
# ---------------------
Kadai4ACM:
  Type: AWS::CertificateManager::Certificate
  Properties:
    DomainName: !Sub '${DomainName}'
    SubjectAlternativeNames:
      - !Sub ${SubjectAlternativeName}
    DomainValidationOptions:
      - DomainName: !Sub '${DomainName}'
        HostedZoneId: !Sub '${HostZoneId}'
    ValidationMethod: DNS
```

3

3: Issue ACM.

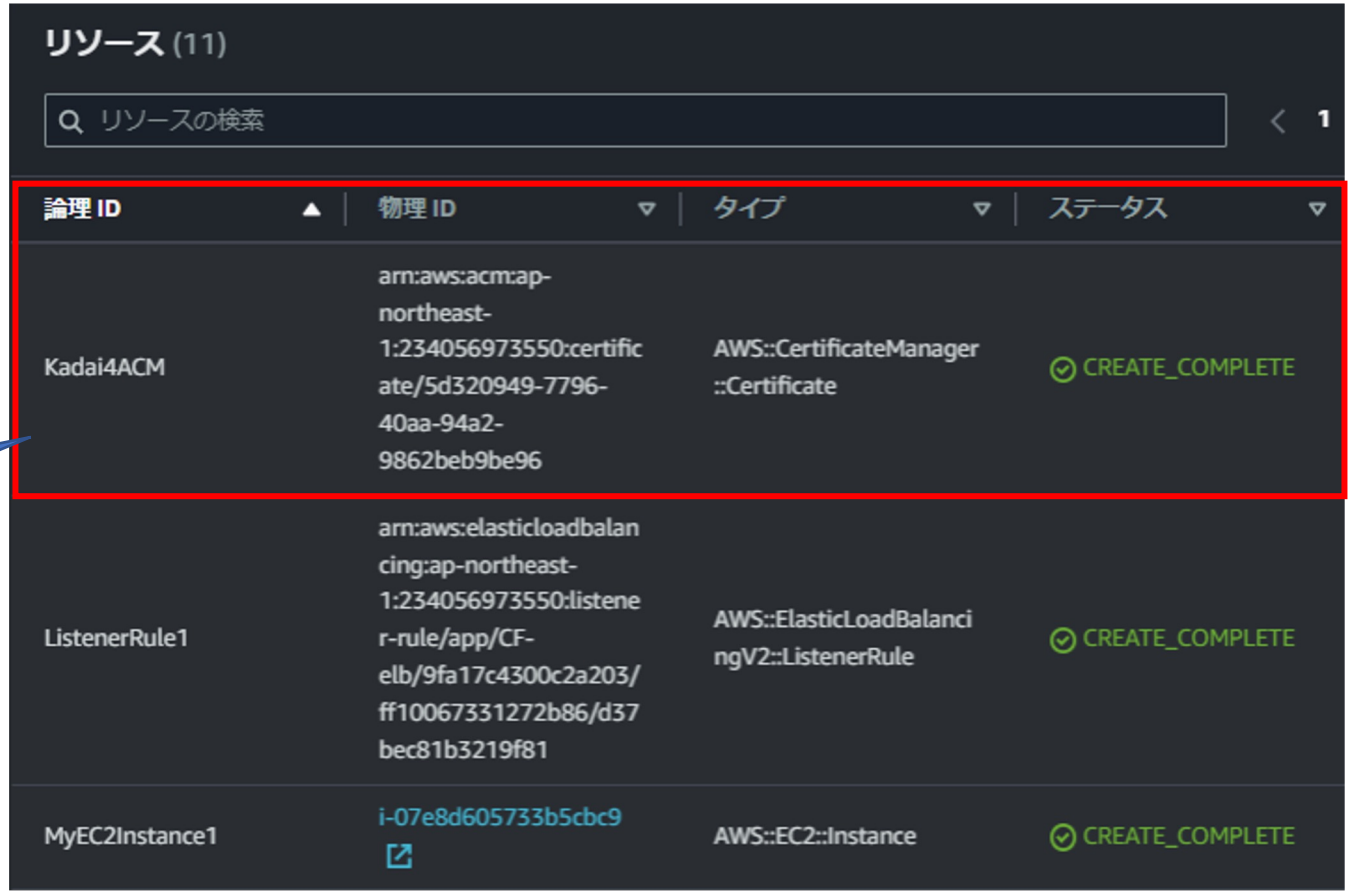
Preparation Step ③:
Issue a certificate in ACM

3. Implementation

Requirement 2: Enable secure encrypted communication using a domain for the web page.

Preparation Step ③:
Issue a certificate in ACM

Using the same YAML file template for EC2, Route53, etc., execute ACM issuance in the same stack.



3. Implementation

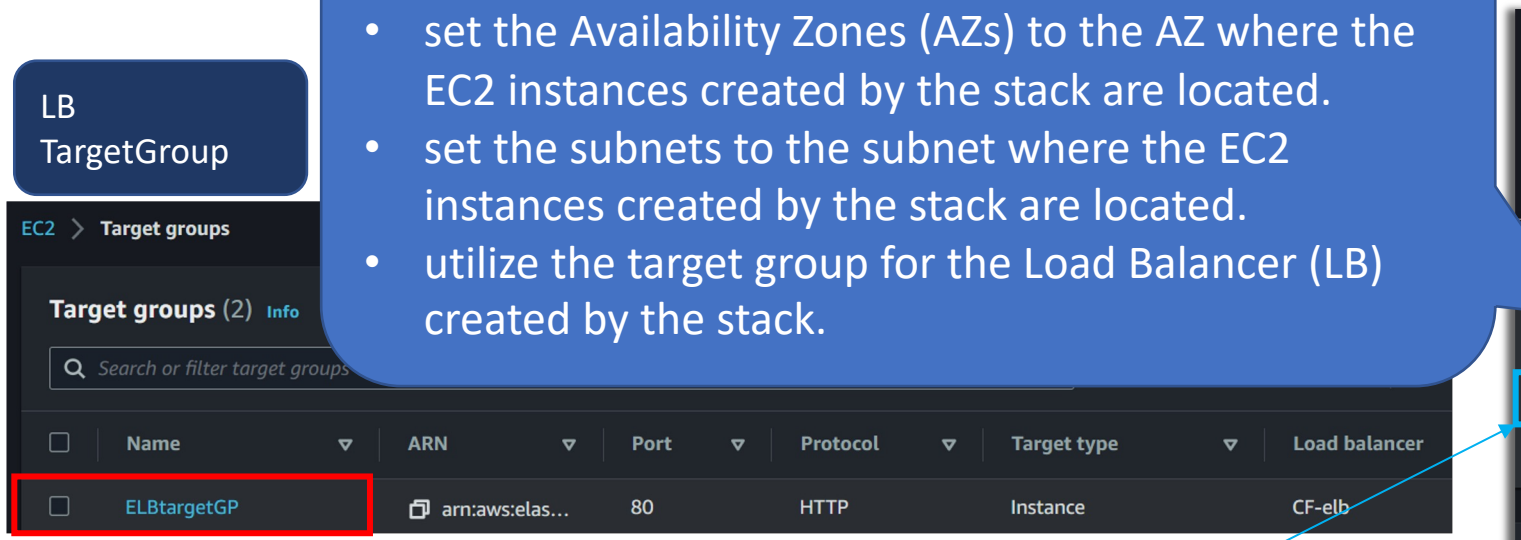
Requirement 3: Enable automatic scaling out when the CPU load exceeds 70% and scaling in when it falls below 20%.

3. Implementation

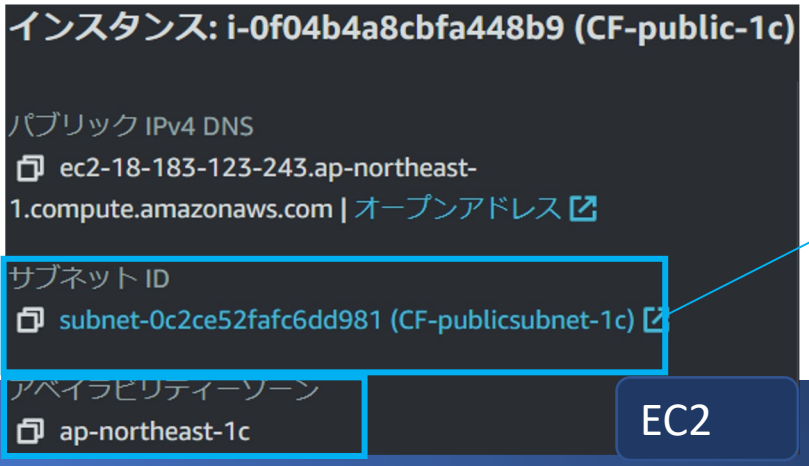
Requirement 3: Enable automatic scaling out when the CPU load exceeds 70% and scaling in when it falls below 20%.

- configure the Auto Scaling Group (ASG):
- set the Availability Zones (AZs) to the AZ where the EC2 instances created by the stack are located.
- set the subnets to the subnet where the EC2 instances created by the stack are located.
- utilize the target group for the Load Balancer (LB) created by the stack.

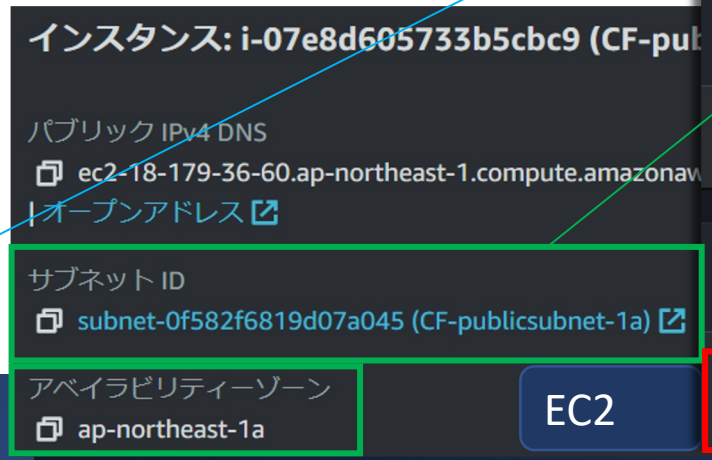
LB
TargetGroup



ASG



EC2



EC2

3. Implementation

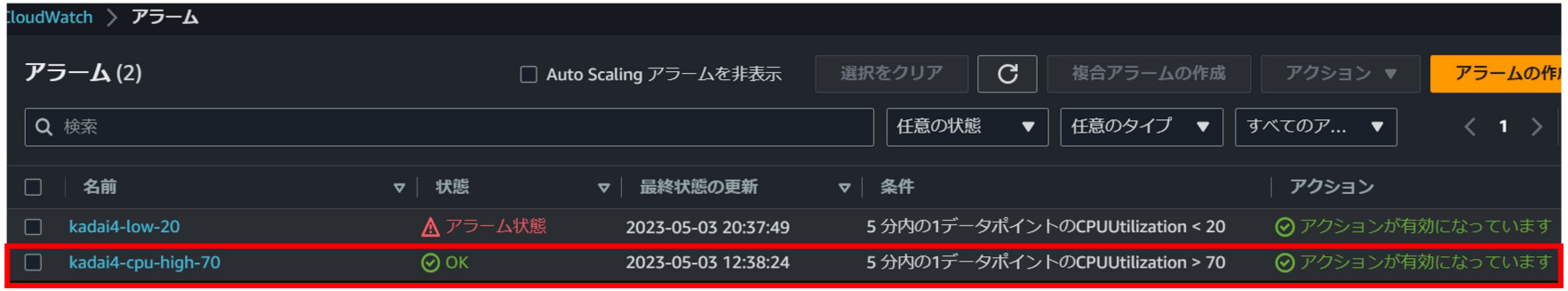
Requirement 3: Enable automatic scaling out when the CPU load exceeds 70% and scaling in when it falls below 20%.

For dynamic scaling policies, set up Simple Scaling Policies:

- when CPU utilization exceeds 70%, add one EC2 instance (Scale Out).
- when CPU utilization falls below 20%, remove one EC2 instance (Scale In).

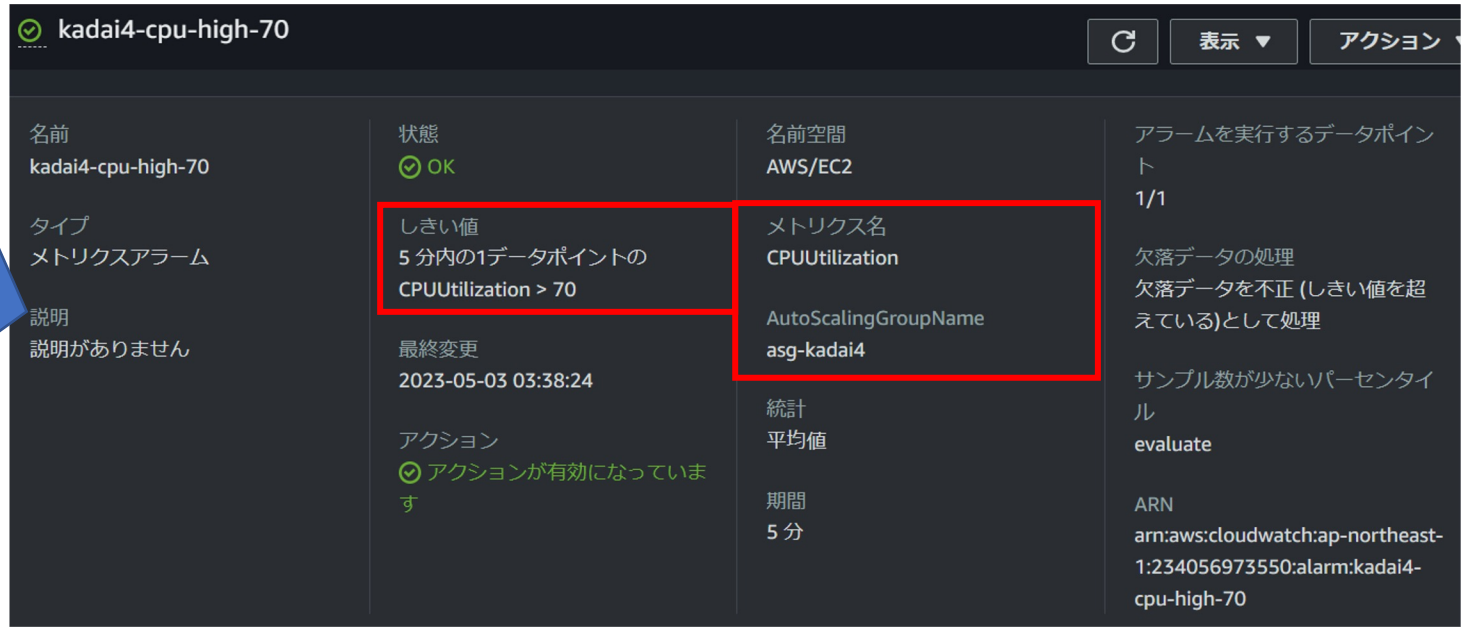
3. Implementation

Requirement 4: Set up alerts to be sent via email to responsible personnel when thresholds are exceeded in monitoring.



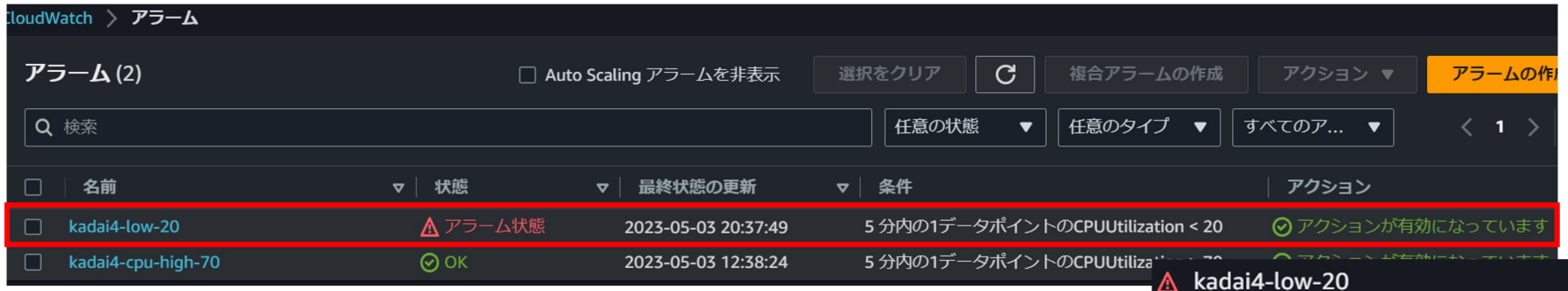
Configure an alert to be triggered when CPU utilization exceeds 70%:

- metric: CPUUtilization
- threshold: CPUUtilization > 70
- associate with ASG



3. Implementation

Requirement 4: Set up alerts to be sent via email to responsible personnel when thresholds are exceeded in monitoring.



Configure an alert to be triggered when CPU utilization falls below 20%:

- metric: CPUUtilization
- threshold: CPUUtilization < 20
- associate with ASG

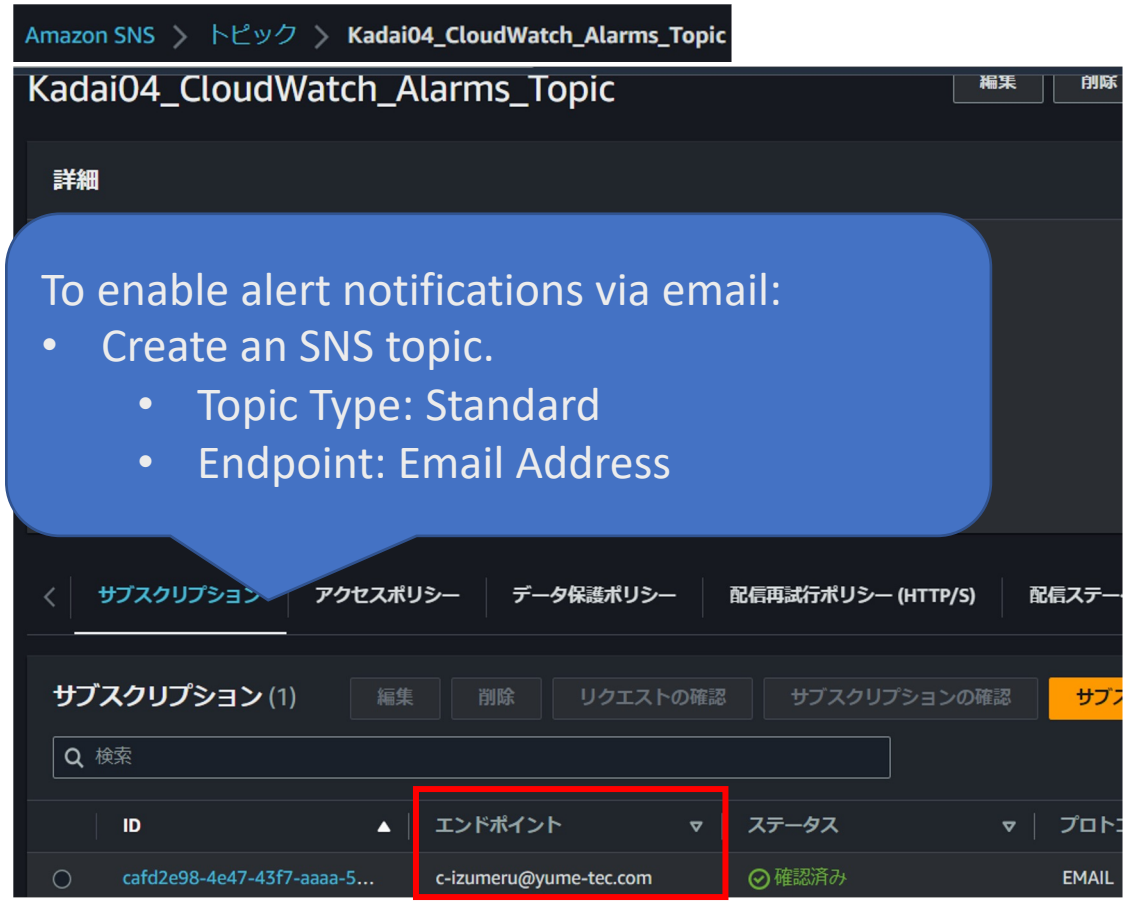
3. Implementation

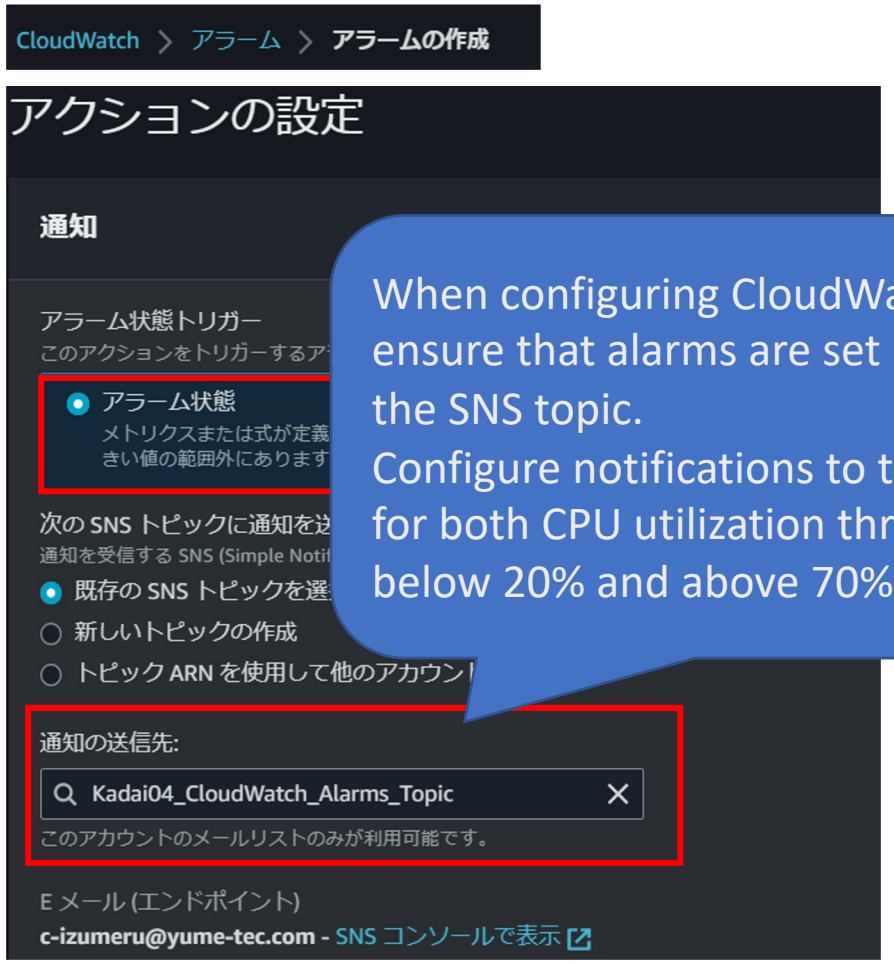
Requirement 4: Set up alerts to be sent via email to responsible personnel when thresholds are exceeded in monitoring.

When setting up the dynamic scaling policies with Simple Scaling Policy in ASG, you can confirm that CloudWatch alarms are configured.

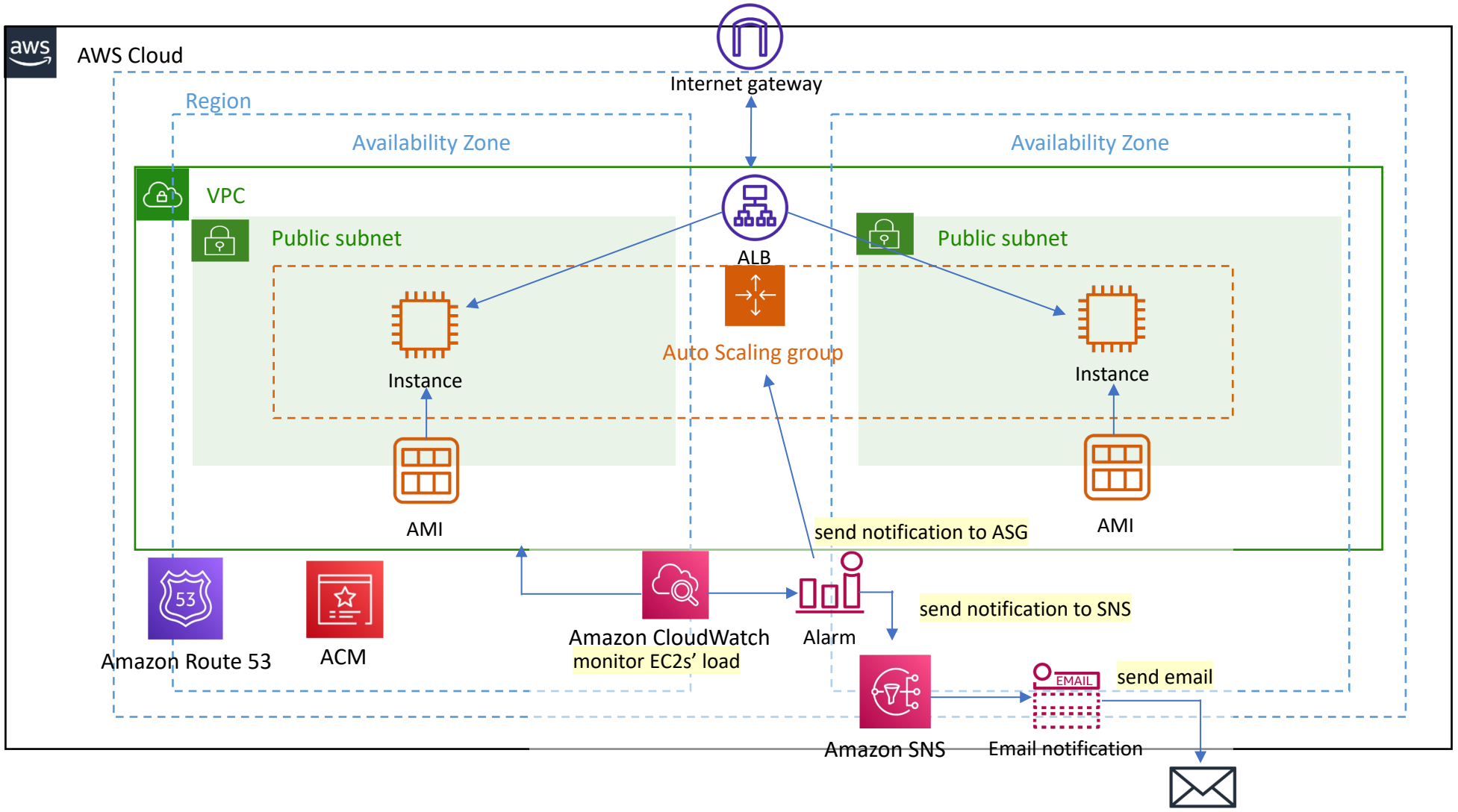
3. Implementation

Requirement 4: Set up alerts to be sent via email to responsible personnel when thresholds are exceeded in monitoring.





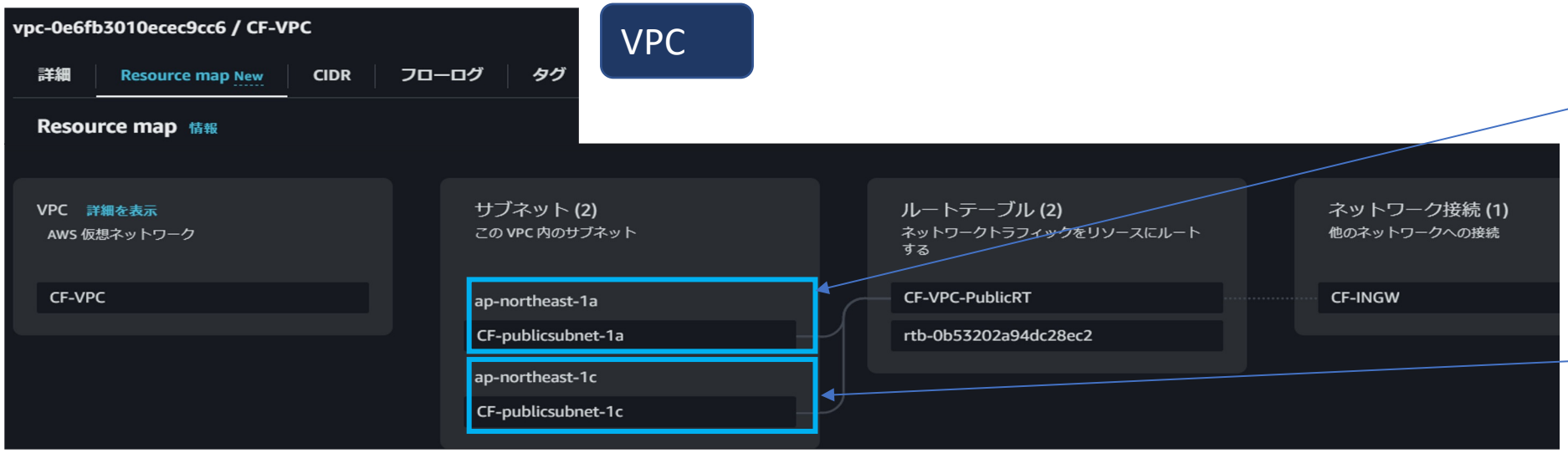
3. Implementation



4. Test Results:

Requirement 1: Automatically build an environment that fulfills the following conditions:

1. Create public subnets in two Availability Zones (AZs).



VPC

```
PublicSubnet1A:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyFirstVPC
    CidrBlock: 10.0.0.0/24
    MapPublicIpOnLaunch: true
    AvailabilityZone: "ap-northeast-1a"
    Tags:
    - Key: Name
      Value: CF-publicsubnet-1a
```

```
PublicSubnet1C:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyFirstVPC
    CidrBlock: 10.0.2.0/24
    MapPublicIpOnLaunch: true
    AvailabilityZone: "ap-northeast-1c"
    Tags:
    - Key: Name
      Value: CF-publicsubnet-1c
```

```
Resources:
  MyFirstVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsSupport: 'true'
      EnableDnsHostnames: 'true'
      InstanceTenancy: default
      Tags:
      - Key: Name
        Value: CF-VPC
```

4. Test Results:

Requirement 1: Automatically build an environment that fulfills the following conditions:
1. Create public subnets in two Availability Zones (AZs).
②-a: Deploy EC2 instances in each public subnet.

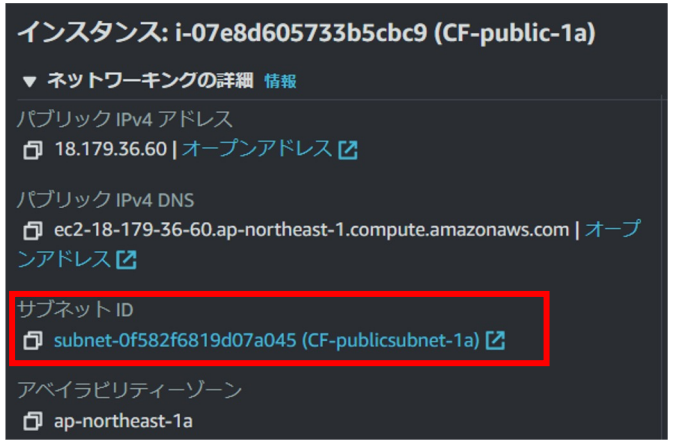
```
PublicSubnet1A:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyFirstVPC
    CidrBlock: 10.0.0.0/24
    MapPublicIpOnLaunch: true
    AvailabilityZone: "ap-northeast-1a"
    Tags:
    - Key: Name
      Value: CF-publicsubnet-1a

PubSubnet1ARouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet1A
    RouteTableId: !Ref PublicRouteTable

PublicSubnet1C:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyFirstVPC
    CidrBlock: 10.0.2.0/24
    MapPublicIpOnLaunch: true
    AvailabilityZone: "ap-northeast-1c"
    Tags:
    - Key: Name
      Value: CF-publicsubnet-1c
```

```
Resources:
  # --------------------
  # EC2 for pub sub 1A
  # --------------------
  MyEC2Instance1:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !Ref EC2AMI
      InstanceType: t2.micro
      SubnetId: !ImportValue VPC-koutiku-PublicSubnet1A
```

```
# --------------------
# EC2 for pub sub 1C
# --------------------
MyEC2Instance2:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: !Ref EC2AMI
    InstanceType: t2.micro
    SubnetId: !ImportValue VPC-koutiku-PublicSubnet1C
```

subnet

The EC2 instances are deployed in two different subnets as specified in the template.



4. Test Results:

Requirement 1: Automatically build an environment that fulfills the following conditions:
1. Create public subnets in two Availability Zones (AZs).
②-a: Deploy EC2 instances in each public subnet.

```
PublicSubnet1A:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyFirstVPC
    CidrBlock: 10.0.0.0/24
    MapPublicIpOnLaunch: true
    AvailabilityZone: "ap-northeast-1a"
    Tags:
    - Key: Name
      Value: CF-publicsubnet-1a

PubSubnet1ARouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet1A
    RouteTableId: !Ref PublicRouteTable

PublicSubnet1C:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyFirstVPC
    CidrBlock: 10.0.2.0/24
    MapPublicIpOnLaunch: true
    AvailabilityZone: "ap-northeast-1c"
    Tags:
    - Key: Name
      Value: CF-publicsubnet-1c
```

```
Resources:
  # --------------------
  # EC2 for pub sub 1A
  # --------------------
  MyEC2Instance1:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !Ref EC2AMI
      InstanceType: t2.micro
      SubnetId: !ImportValue VPC-koutiku-PublicSubnet1A
```

```
# --------------------
# EC2 for pub sub 1C
# --------------------
MyEC2Instance2:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: !Ref EC2AMI
    InstanceType: t2.micro
    SubnetId: !ImportValue VPC-koutiku-PublicSubnet1C
```

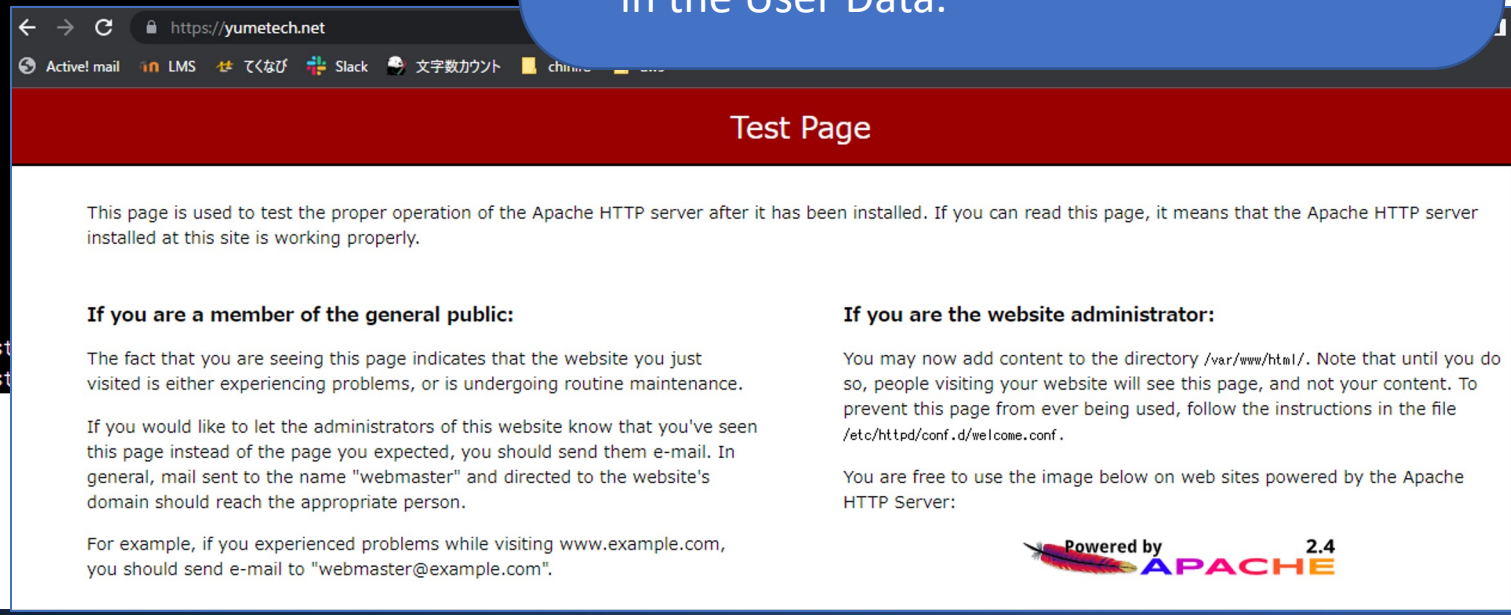
The EC2 instances are launched in two different AZs as specified in the template.

4. Test Results:

Requirement 1: Automatically build an environment that fulfills the following conditions:
1. Create public subnets in two Availability Zones (AZs).
②-b: Use Apache as the web server

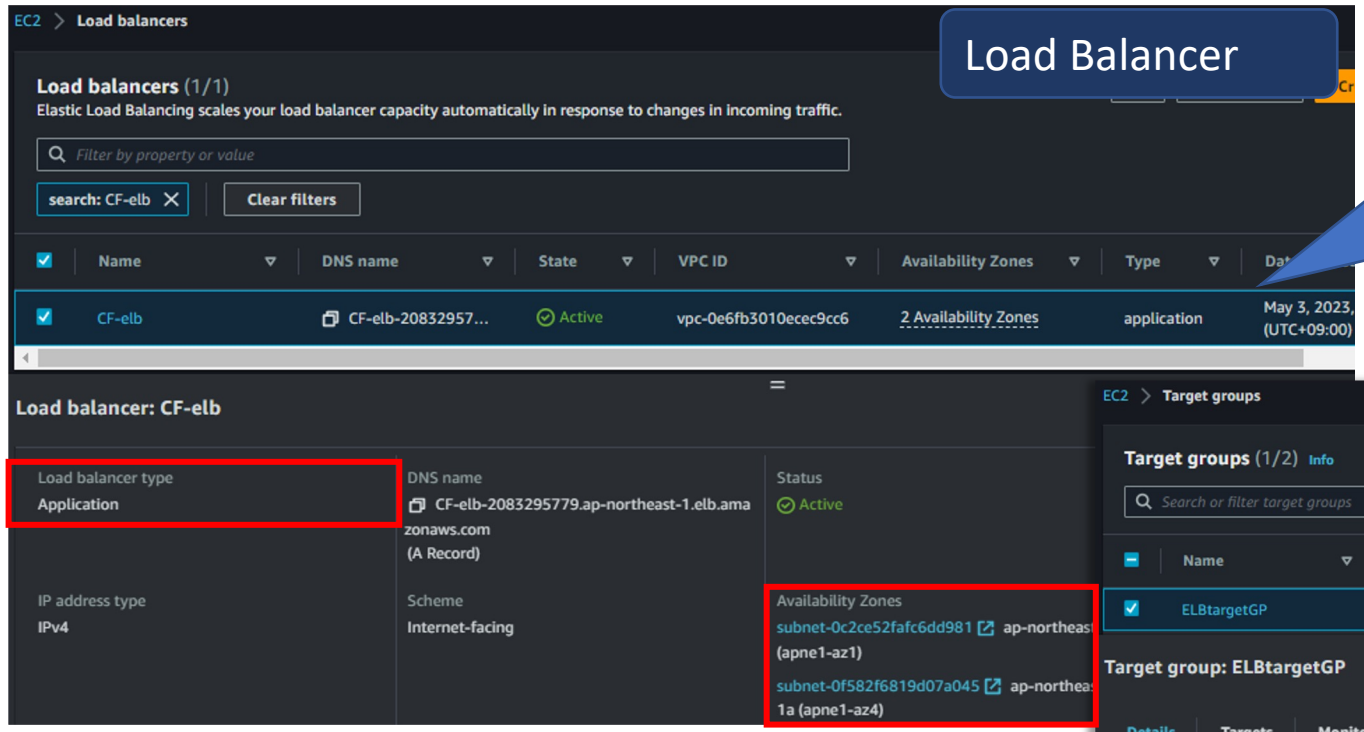
```
https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-0-197 ~]$ sudo service httpd status
Redirecting to /bin/systemctl status httpd.service
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2023-05-03 00:44:44 UTC; 1 day 3h ago
     Docs: man:httpd.service(8)
 Main PID: 6294 (httpd)
   Status: "Total requests: 6816; Idle/Busy workers 100/0;Requests/sec: 0.0696; Bytes served/sec
   CGroup: /system.slice/httpd.service
           ├─6294 /usr/sbin/httpd -DFOREGROUND
           ├─6295 /usr/sbin/httpd -DFOREGROUND
           ├─6296 /usr/sbin/httpd -DFOREGROUND
           ├─6297 /usr/sbin/httpd -DFOREGROUND
           ├─6298 /usr/sbin/httpd -DFOREGROUND
           ├─6299 /usr/sbin/httpd -DFOREGROUND
           ├─6378 /usr/sbin/httpd -DFOREGROUND
           ├─6384 /usr/sbin/httpd -DFOREGROUND
           ├─6385 /usr/sbin/httpd -DFOREGROUND
           └─9076 /usr/sbin/httpd -DFOREGROUND

May 03 00:44:43 ip-10-0-0-197.ap-northeast-1.compute.internal syst
May 03 00:44:44 ip-10-0-0-197.ap-northeast-1.compute.internal syst
```

- connected to the EC2 instance via EC2 Instance Connect
- executed the sudo service httpd status command.
- Apache is installed as per the script written in the User Data.



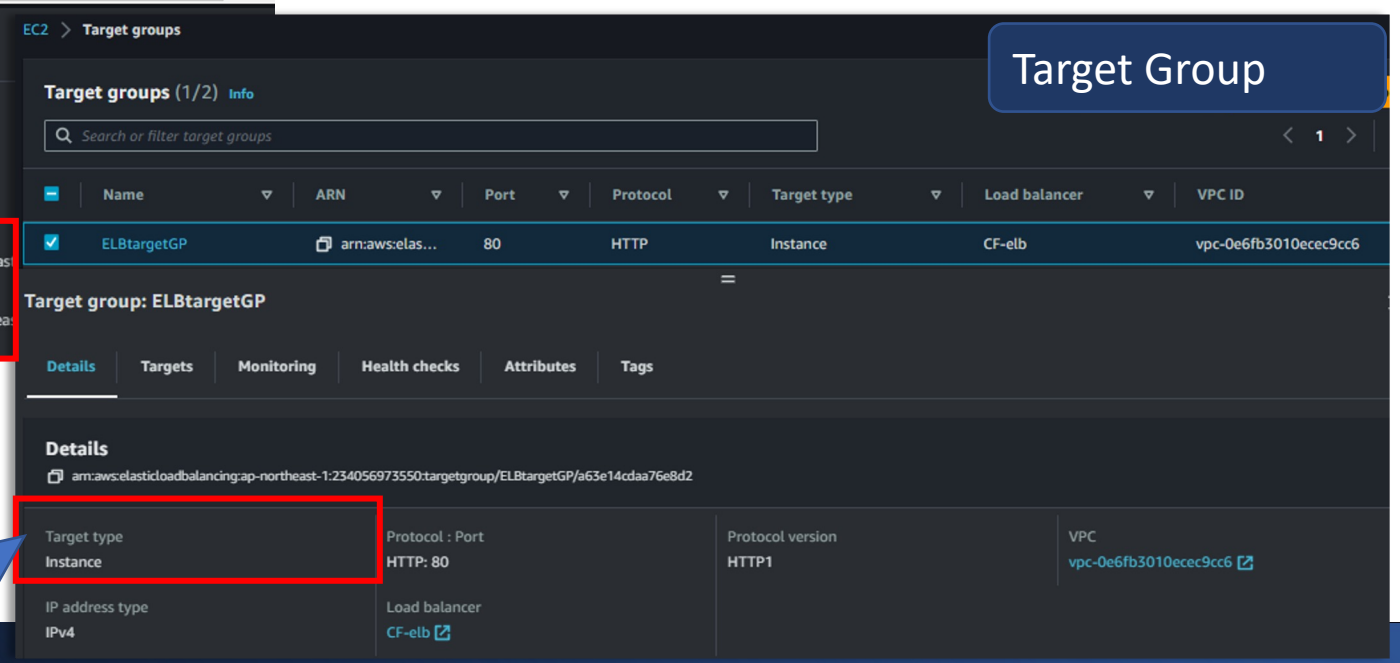
4. Test Results:

Requirement 1: Automatically build an environment that fulfills the following conditions:
1. Create public subnets in two Availability Zones (AZs).
③: Use ALB and configure each EC2 instance in the target group.



Load Balancer

- Load Balancer Type: Application
- AZ: Configured to span across the AZs specified in the template.

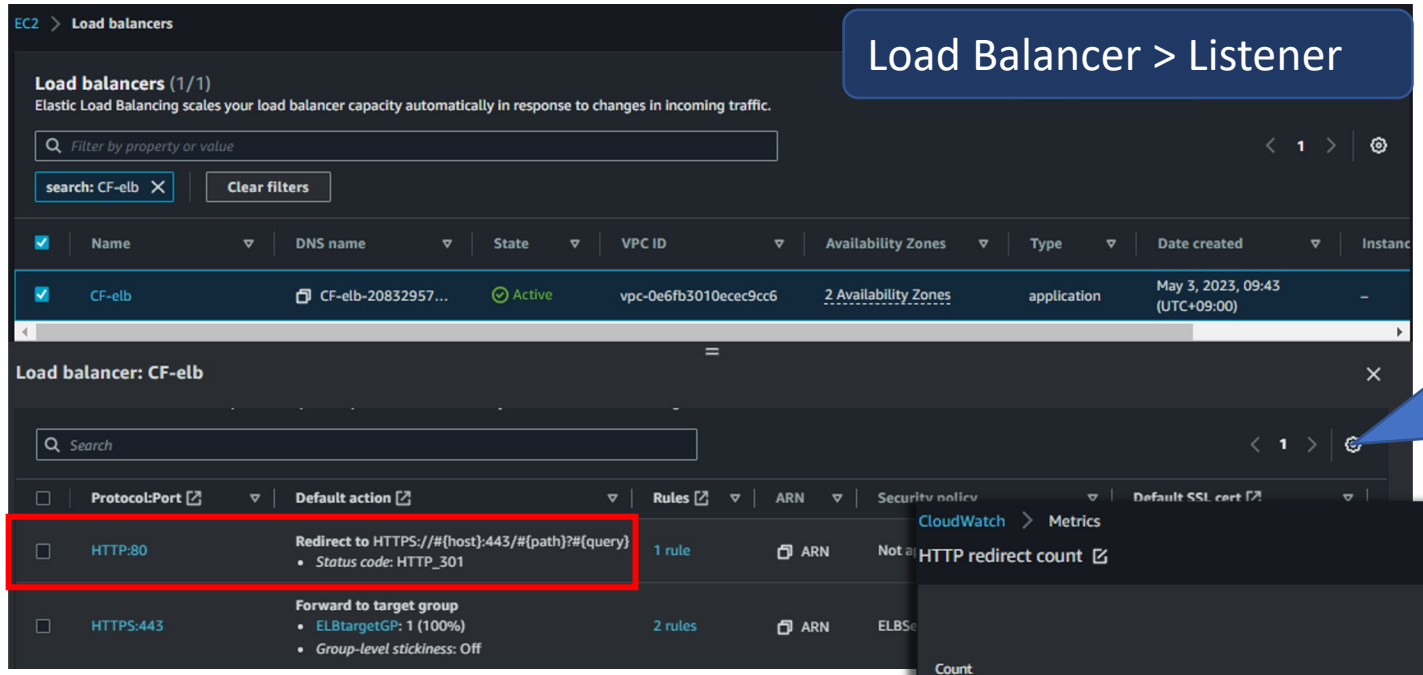


Target Group

- Target Group set to Instance (EC2).

4. Test Results:

Requirement 2: Enable secure encrypted communication using a domain for the web page.



Load Balancer > Listener

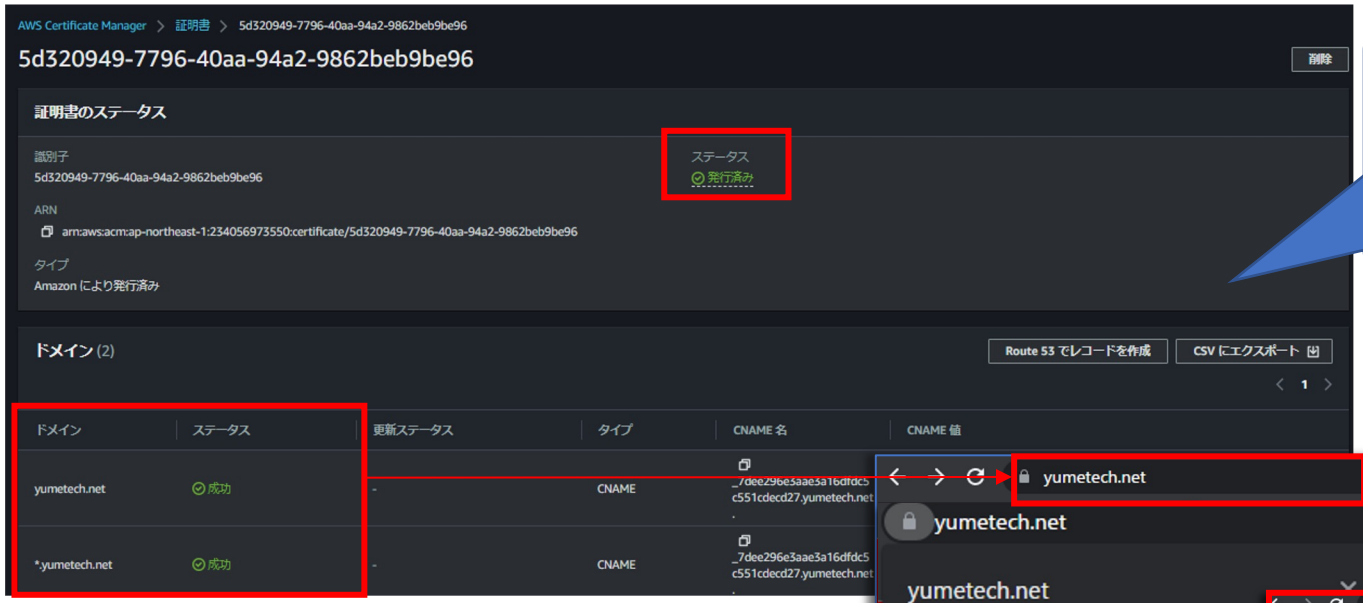
The Default Action for HTTP port 80 is set to redirect to HTTPS instead of forwarding to the target group.

CloudWatch

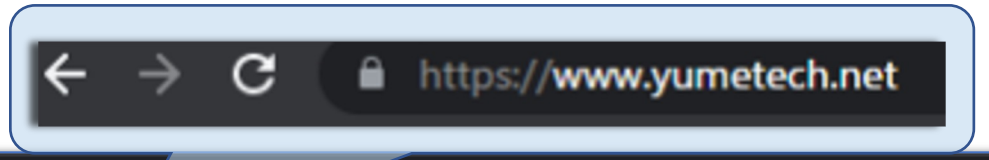
Verified from CloudWatch metrics that redirection to HTTPS is occurring.

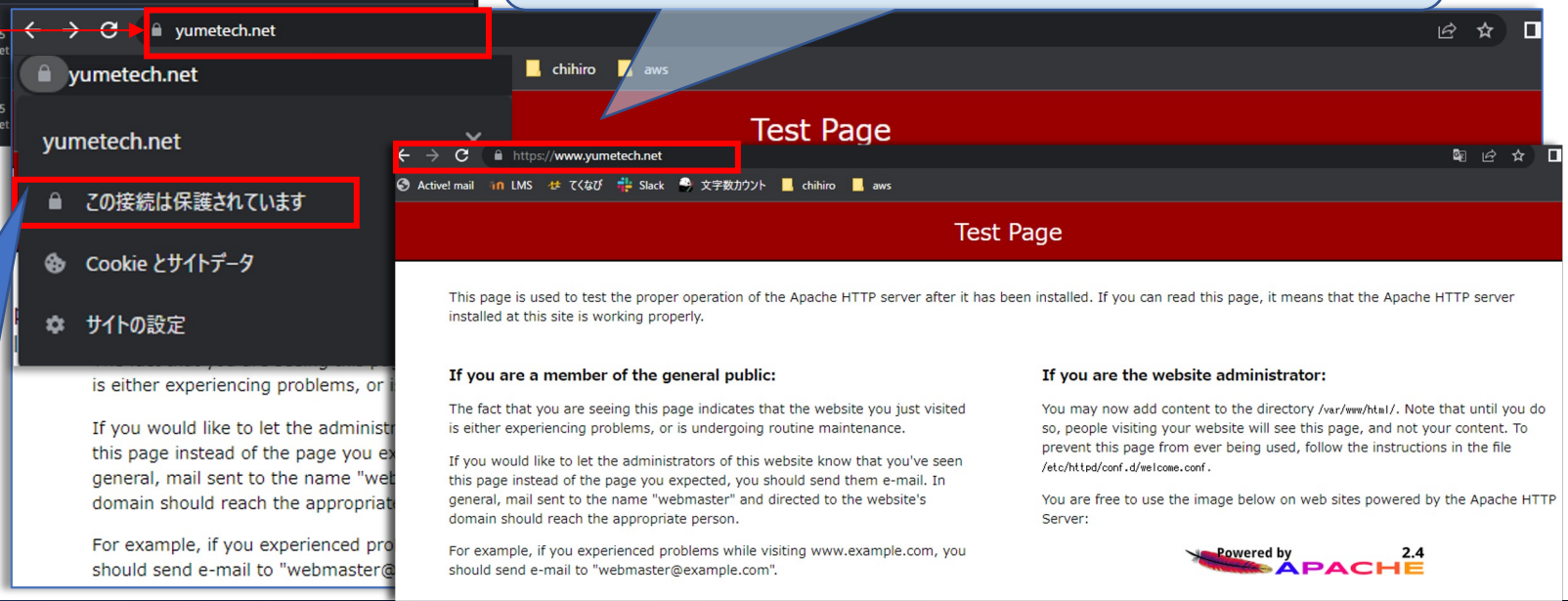
4. Test Results:

Requirement 2: Enable secure encrypted communication using a domain for the web page.



- An ACM certificate has been issued
- a CNAME for DNS validation has been set up.





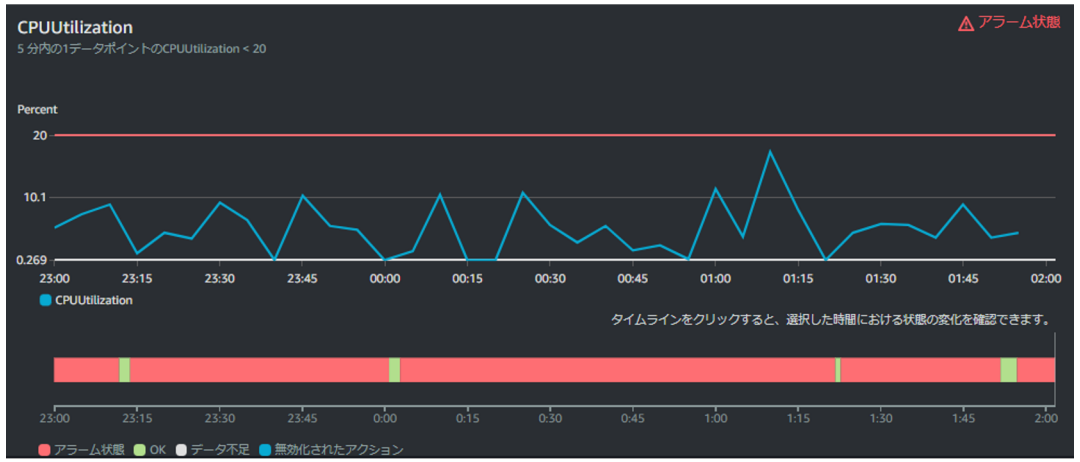
- When accessing the DNS-validated domain name, we can confirm that it’s an HTTPS connection.
- The subdomain www.yumetech.net is also configured for HTTPS connection.

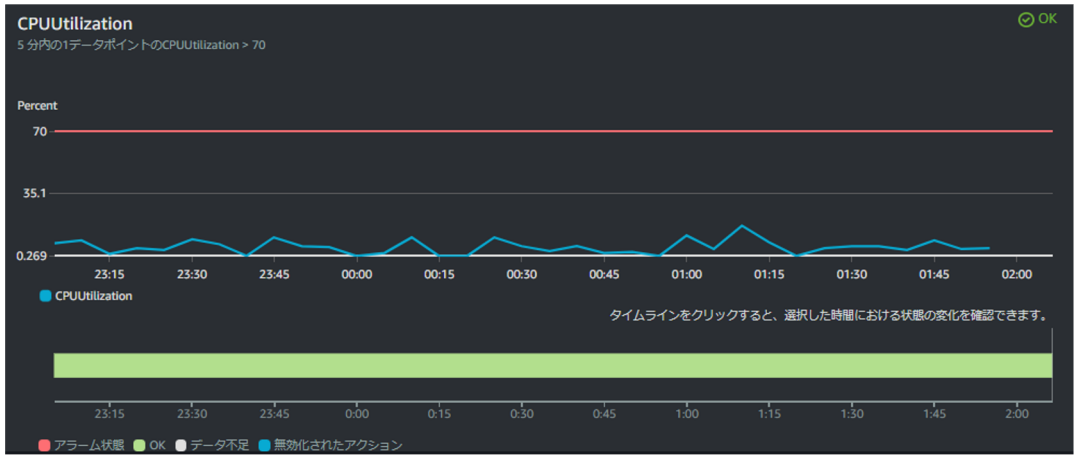
4. Test Results:

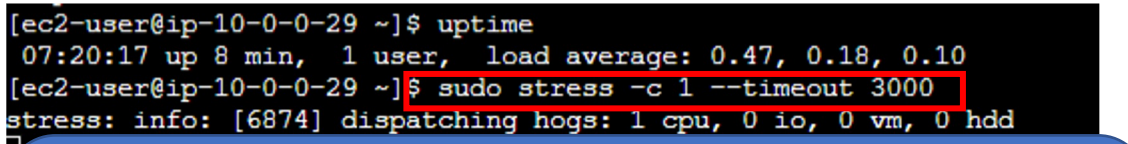
Requirement 3: make EC2 automatically scale out when the CPU load exceeds 70% and scale in when it falls below 20%.

Before applying any load

No instances have experienced CPU loads above 20% or 70%.





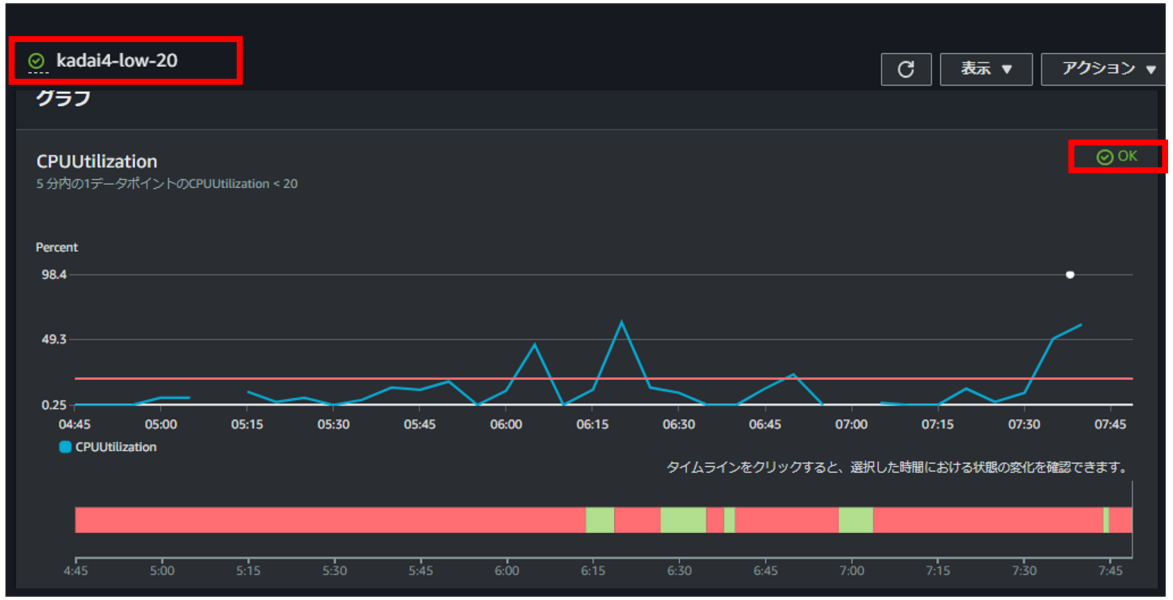


Applied load to these EC2 instances using the stress command.
sudo amazon-linux-extras install epel –y
sudo yum install stress –y

4. Test Results:

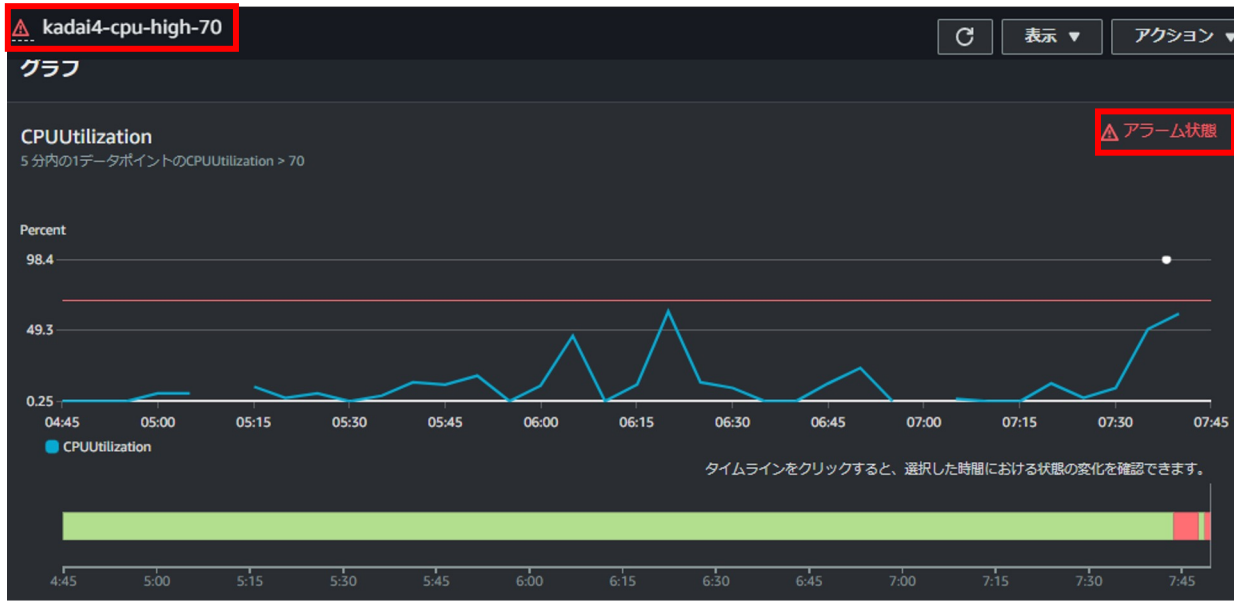
Requirement 3: make EC2 automatically scale out when the CPU load exceeds 70% and scale in when it falls below 20%.

Status while applying load



CPU load exceeds 20%, causing the CloudWatch alarm set at CPU 20% to return to the "OK" state.

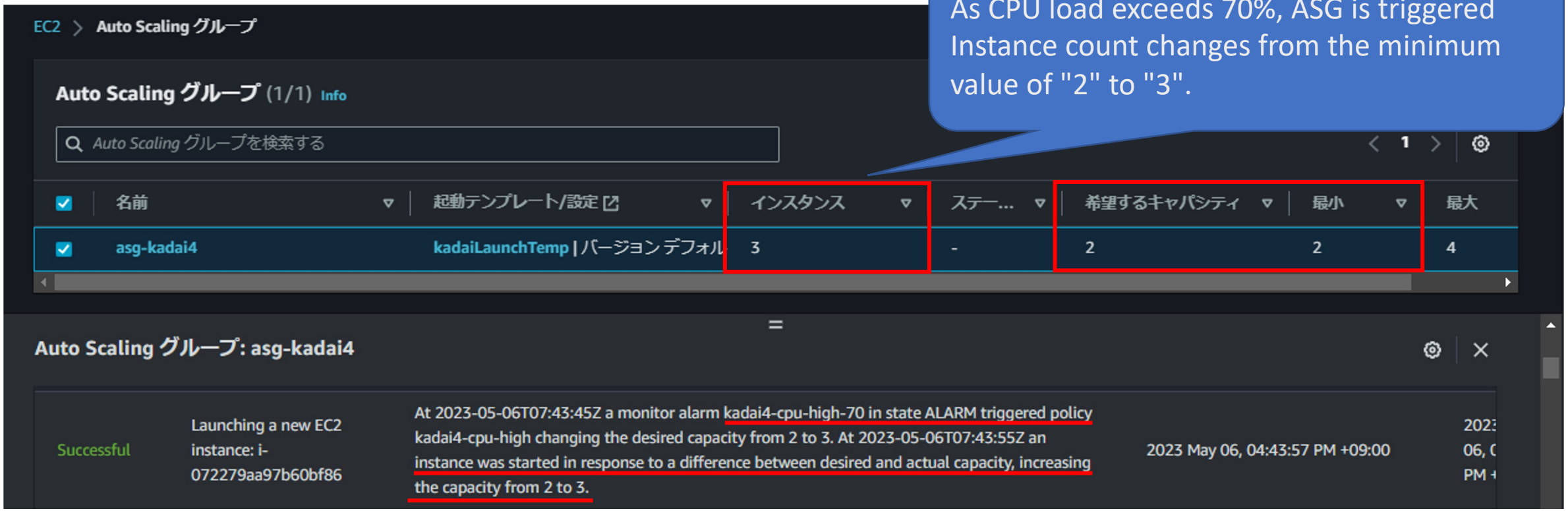
CPU load exceeds 70%, triggering the CloudWatch alarm set at CPU 70% to enter the "Alarm" state.



4. Test Results:

Requirement 3: make EC2 automatically scale out when the CPU load exceeds 70% and scale in when it falls below 20%.

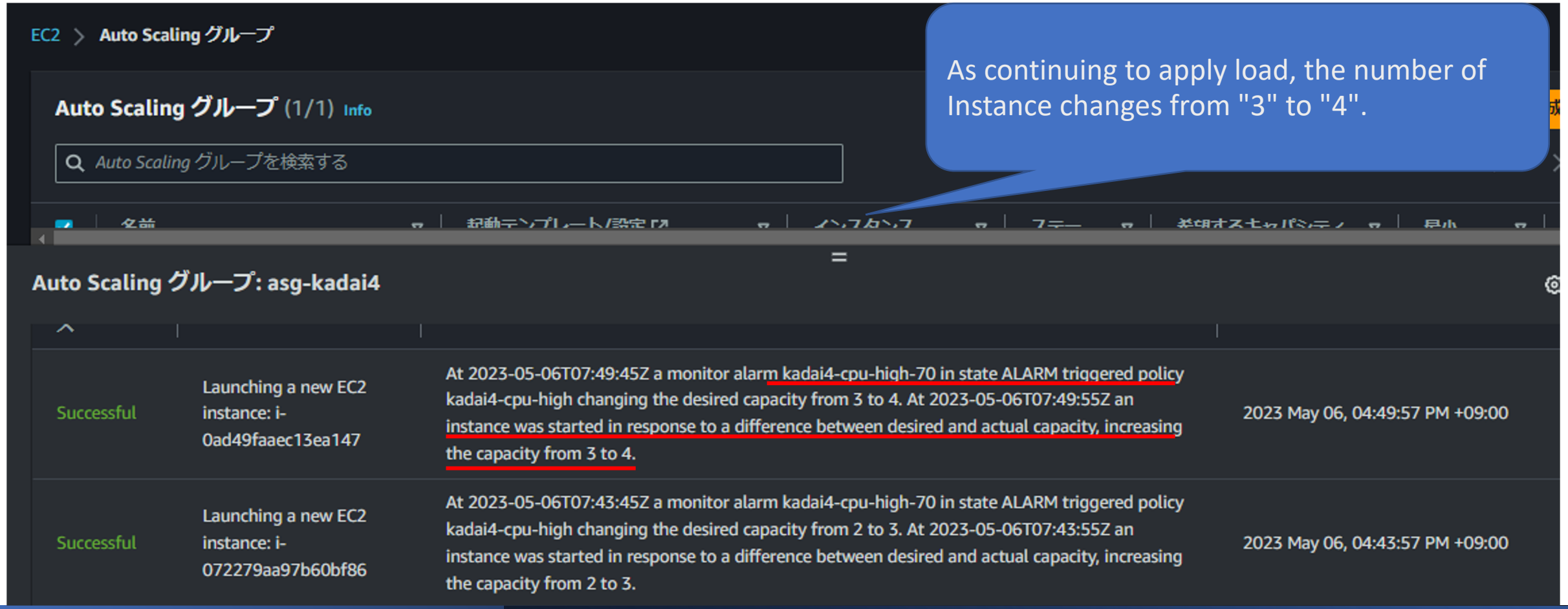
scaling out



4. Test Results:

Requirement 3: make EC2 automatically scale out when the CPU load exceeds 70% and scale in when it falls below 20%.

scaling out



As continuing to apply load, the number of Instance changes from "3" to "4".

4. Test Results:

Requirement 3: make EC2 automatically scale out when the CPU load exceeds 70% and scale in when it falls below 20%.

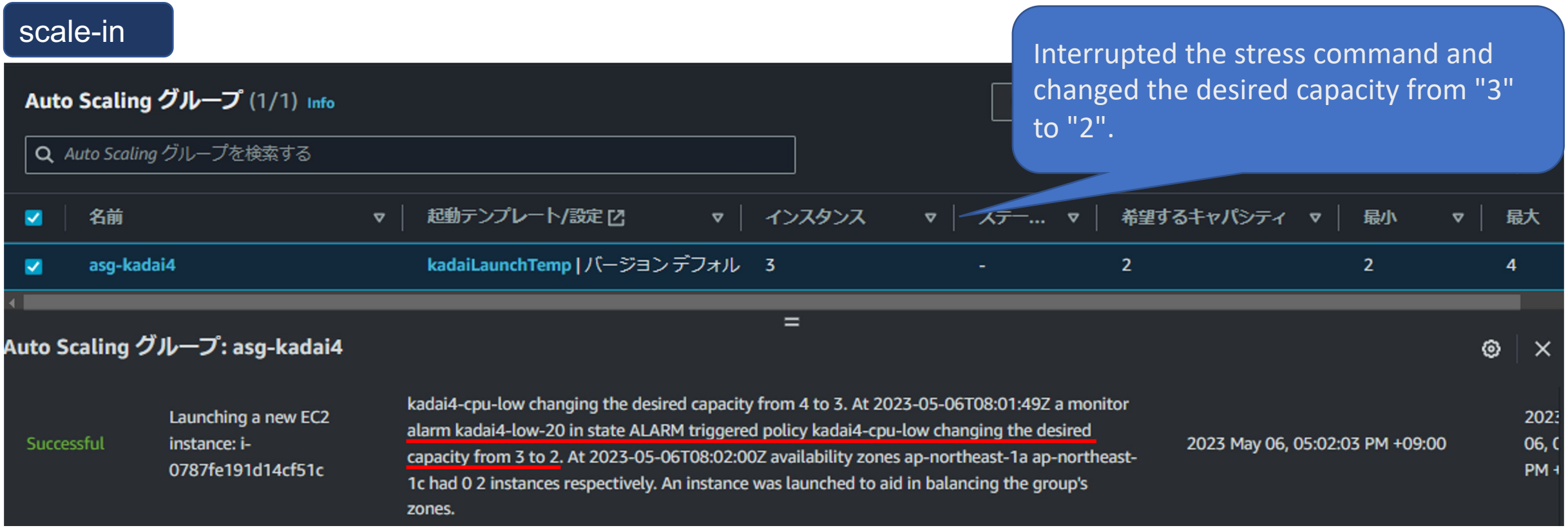
scale-out

2 instances created from the stack（CF-public-1a, CF-public-1c）
+ 4 instances scaled out by ASG（kadai4-asg x4）= 6

4. Test Results:

Requirement 3: make EC2 automatically scale out when the CPU load exceeds 70% and scale in when it falls below 20%.

scale-in



4. Test Results:

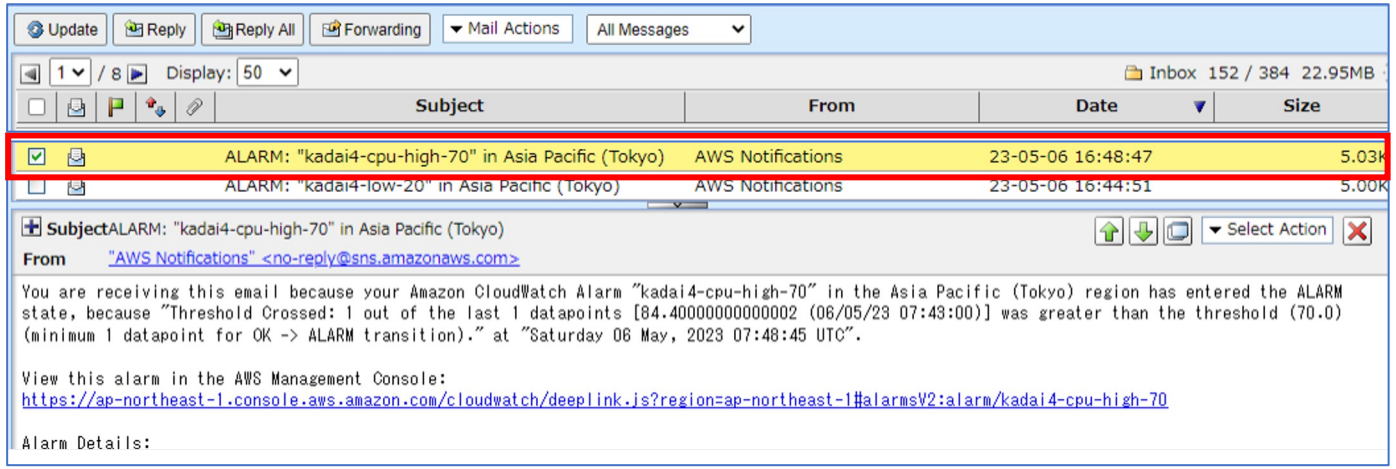
Requirement 3: make EC2 automatically scale out when the CPU load exceeds 70% and scale in when it falls below 20%.

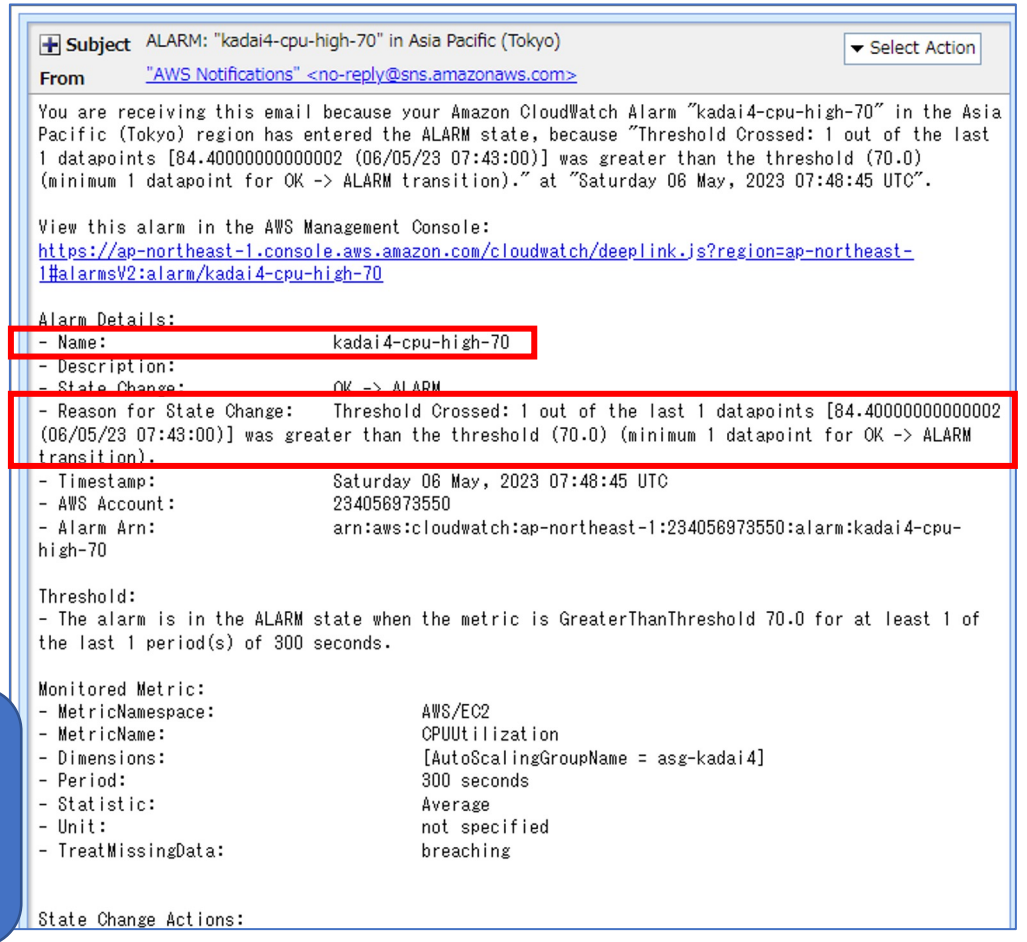
Scale-in

2 instances created from the stack （CF-public-1a, CF-public-1c）
+ 2 instances scaled in by ASG（kadai4-asg x2）＝ 4

4. Test Results:

Requirement 4: Set up alerts to be sent via email to responsible personnel when thresholds are exceeded in monitoring.

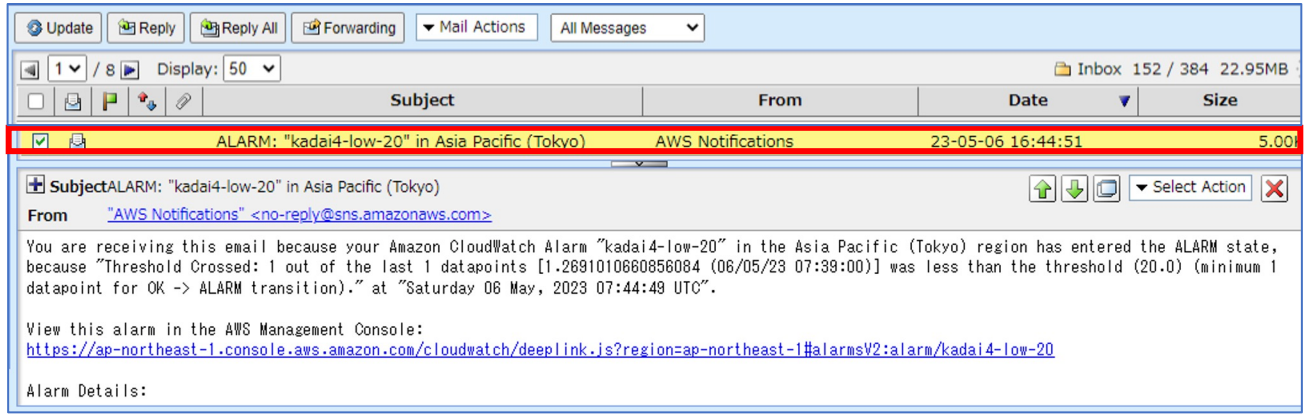


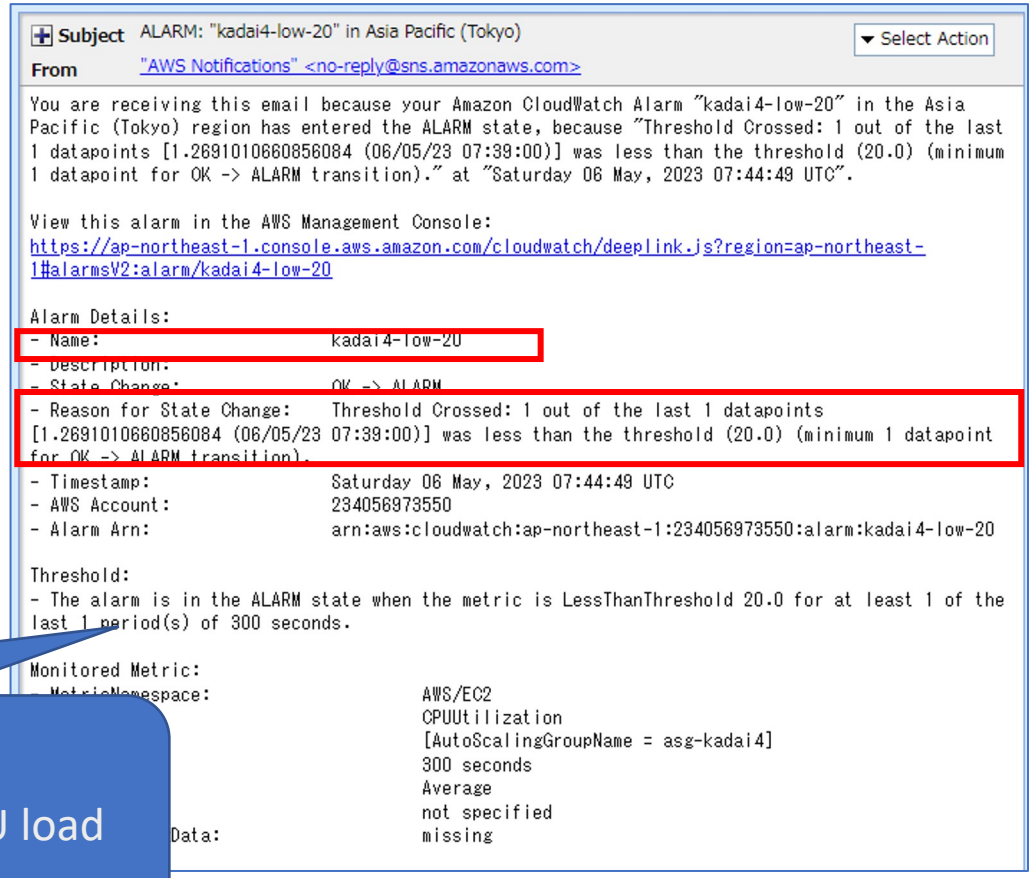


The responsible person's email address, registered as SNS Topics, received notifications for CloudWatch alarms triggered when CPU load exceeded 70%.

4. Test Results:

Requirement 4: Set up alerts to be sent via email to responsible personnel when thresholds are exceeded in monitoring.





The responsible person's email address, registered as SNS Topics, received notifications for CloudWatch alarms triggered when CPU load fell below 20%.

Fin.