テスト(課題)

キム テフン

jkdnekki@naver.com

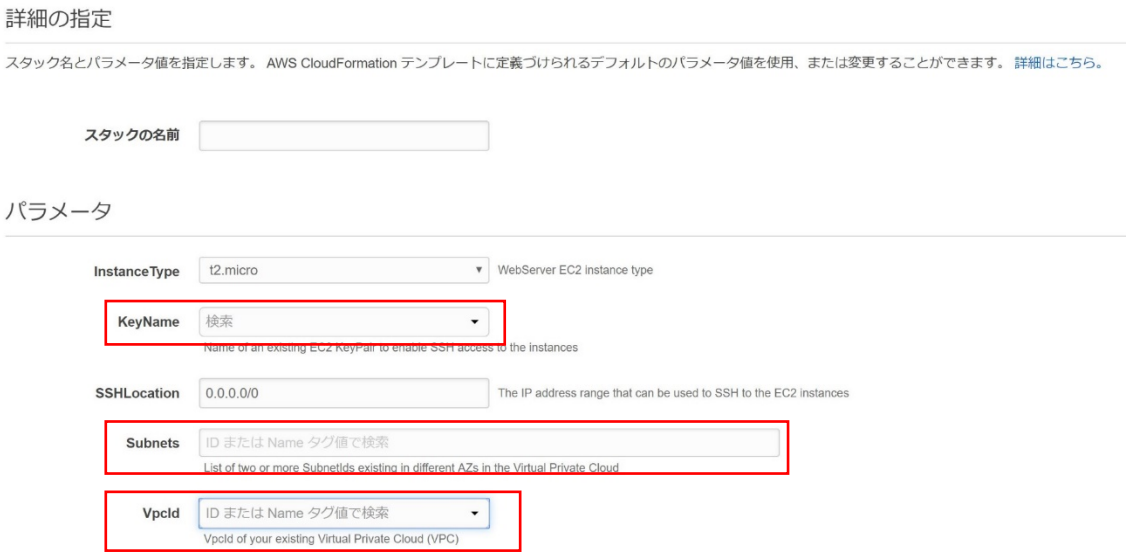
目次

環境構築について

下記のテンプレートを使うためには**事前準備**が必要となります。

https://s3-ap-northeast-1.amazonaws.com/cm-ope-exam-prod/CM-Ope-CFnTemplates.yaml?versionId=null

・VPC Subnetはデフォルトでも可能ですが、勉強を含めてテンプレート作成し、それを使ってみました。最後の【その他】のページに載せましたのでご参考ください。



事前準備

・キーペア
　ネットワーク＆セキュリティ→キーペア
・AWS CLI Installと　AWS Configureの設定
・VPCとsubnetを作成(その他のページ参考、必須X)
・JQコマンドインストール(その他のページ参考、必須X)

■ 課題

・TEST 1. どちらか一方の EC2 インスタンスを Stopし、その後 Start

・TEST 2. 障害原因の調査及び回避対策

TEST 1. どちらか一方の EC2 インスタンスを Stop し、その後 Start

一方のEC2インスタンスをstopする前に、Auto Scalingグループに設定されている インスタンスIDを検索します。
以下のようにELBに繋がれているインスタンスIDの確認ができます。

```
$ aws autoscaling describe-auto-scaling-instances --query 'AutoScalingInstances[].InstanceId'
[
    "i-059e6a5fcc14c5e3b",
    "i-06bff9f37aa74858f"
]
```

事前にインストールしたJQ で、インスタンスのHealth Statusをチェックすると正常に動いていることが確認できます。

```
$ aws autoscaling describe-auto-scaling-instances | jq '.AutoScalingInstances[] |{InstanceId, HealthStatus}'
{
  "InstanceId": "i-06bff9f37aa74858f",
  "HealthStatus": "HEALTHY"
}
{
  "InstanceId": " i-059e6a5fcc14c5e3b ",
  "HealthStatus": "HEALTHY"
}
```

＊赤色のインスタンスを以下のAWS CLIコマンドで停止させます。

```
$ aws ec2 stop-instances --instance-ids i-059e6a5fcc14c5e3b
```

←――― インスタンスを Stop!

Stop後、Auto Scalingをチェックしてみると 、stop前にHealthyだったステータスがstop後にUnhealthyになることを確認できます。

```
$ aws autoscaling describe-auto-scaling-groups
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "ELBStack0119-WebServerGroup-1ITTZ6D129R3L",
            "AutoScalingGroupARN": "arn:aws:autoscaling:ap-northeast-1:115757856347:autoScalingGroup:228ed108-8589-4ccc-
bfaa-610dd904ec76:autoScalingGroupName/ELBStack0119-WebServerGroup-1ITTZ6D129R3L",
            "LaunchConfigurationName": "ELBStack0119-LaunchConfig-1X98XPEQA0LRM",
            "MinSize": 2,
            "MaxSize": 2,
            "DesiredCapacity": 2,
            "DefaultCooldown": 300,
            "AvailabilityZones": [
                "ap-northeast-1a",
                "ap-northeast-1c"
            ],
            "LoadBalancerNames": [],
            "TargetGroupARNs": [
                "arn:aws:elasticloadbalancing:ap-northeast-1:115757856347:targetgroup/ELBSt-ALBTa-
1D97IKGWMHXXO/f8f8e03f02671681"
            ],
            "HealthCheckType": "EC2",
            "HealthCheckGracePeriod": 0,
            "Instances": [
                {
                    "InstanceId": "i-059e6a5fcc14c5e3b",
                    "InstanceType": "t2.micro",
                    "AvailabilityZone": "ap-northeast-1a",
                    "LifecycleState": "InService",
                    "HealthStatus": "Healthy",
                    "LaunchConfigurationName": "ELBStack0119-LaunchConfig-1X98XPEQA0LRM",
                    "ProtectedFromScaleIn": false
                },
                {
                    "InstanceId": "i-06bff9f37aa74858f",
                    "InstanceType": "t2.micro",
                    "AvailabilityZone": "ap-northeast-1c",
                    "LifecycleState": "InService",
                    "HealthStatus": "Healthy",
                    "LaunchConfigurationName": "ELBStack0119-LaunchConfig-1X98XPEQA0LRM",
                    "ProtectedFromScaleIn": false
                }
            ],
            省略～
```

```
$aws autoscaling describe-auto-scaling-groups
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "ELBStack0119-WebServerGroup-1ITTZ6D129R3L",
            "AutoScalingGroupARN": "arn:aws:autoscaling:ap-northeast-1:115757856347:autoScalingGroup:228ed108-8589-4ccc-bfaa-
610dd904ec76:autoScalingGroupName/ELBStack0119-WebServerGroup-1ITTZ6D129R3L",
            "LaunchConfigurationName": "ELBStack0119-LaunchConfig-1X98XPEQA0LRM",
            "MinSize": 2,
            "MaxSize": 2,
            "DesiredCapacity": 2,
            "DefaultCooldown": 300,
            "AvailabilityZones": [
                "ap-northeast-1a",
                "ap-northeast-1c"
            ],
            "LoadBalancerNames": [],
            "TargetGroupARNs": [
                "arn:aws:elasticloadbalancing:ap-northeast-1:115757856347:targetgroup/ELBSt-ALBTa-
1D97IKGWMHXXO/f8f8e03f02671681"
            ],
            "HealthCheckType": "EC2",
            "HealthCheckGracePeriod": 0,
            "Instances": [
                {
                    "InstanceId": "i-059e6a5fcc14c5e3b",
                    "InstanceType": "t2.micro",
                    "AvailabilityZone": "ap-northeast-1a",
                    "LifecycleState": "Terminating",
                    "HealthStatus": "Unhealthy",
                    "LaunchConfigurationName": "ELBStack0119-LaunchConfig-1X98XPEQA0LRM",
                    "ProtectedFromScaleIn": false
                },
                {
                    "InstanceId": "i-06bff9f37aa74858f",
                    "InstanceType": "t2.micro",
                    "AvailabilityZone": "ap-northeast-1c",
                    "LifecycleState": "InService",
                    "HealthStatus": "Healthy",
                    "LaunchConfigurationName": "ELBStack0119-LaunchConfig-1X98XPEQA0LRM",
                    "ProtectedFromScaleIn": false
                }
            ],
            省略～
```

*原因:最少1つのインスタンスが必要なところが、2以下になった場合、AutoScalingがUnhealthyと判断します。
MinSizeの設定が2になっているので停止させたインスタンスはTerminateされ、新しいインスタンスが生成されます。

AWS CLIでAuto Scalingグループの状況確認すると
以下のようにELBに新しいインスタンス(緑色)が作成されていることが確認できます。

```
$ aws autoscaling describe-auto-scaling-instances --query 'AutoScalingInstances[].InstanceId'
[
    "i-059e6a5fcc14c5e3b",
    "i-06bff9f37aa74858f",
    "i-0bbdc09f9a55c37de"
]
```

*Stopになっているインスタンスを再びstartしてもAutoScalingにより削除されているので、以下のようにインスタンスが存在しないと言うメッセージが出ます。

```
$ aws ec2 start-instances --instance-ids i-059e6a5fcc14c5e3b

An error occurred (InvalidInstanceID.NotFound) when calling the StartInstances
operation: The instance ID 'i-059e6a5fcc14c5e3b' does not exist
```

←――――インスタンスを Start!

*参考:stopコマンドを叩いた直後に(stopping中)startコマンドを叩くと「pending→InServiceになって一時的にInServiceの状態になりますが,
少し時間がたつとAutoScalingより削除されます。

既存のインスタンス(i-059e6a5fcc14c5e3b)が削除されても、ELB経由で問題なく接続は出来るのが確認できました。

```
$ aws elbv2 describe-load-balancers --query "LoadBalancers[].[LoadBalancerName,DNSName,LoadBalancerArn]"
[
    [
        "ELBSt-Appli-7Q2M2W6ZVQ9U",
        "ELBSt-Appli-7Q2M2W6ZVQ9U-1141534294.ap-northeast-1.elb.amazonaws.com",
        "arn:aws:elasticloadbalancing:ap-northeast-1:115757856347:loadbalancer/app/ELBSt-Appli-7Q2M2W6ZVQ9U/ba5e5816edcbfe52"
    ]
]

$ curl ELBSt-Appli-7Q2M2W6ZVQ9U-1141534294.ap-northeast-1.elb.amazonaws.com
<h3>You have successfully launched the Classmethod Operation Team AWS CloudFormation sample</h3>
```

再びAWS CLIでAuto Scalingグループの状況再確認すると
以下のようにHealth stausが正常になっていることを確認できました。

```
$ aws autoscaling describe-auto-scaling-instances | jq '.AutoScalingInstances[] |{InstanceId, HealthStatus}'
{
  "InstanceId": "i-06bff9f37aa74858f",
  "HealthStatus": "HEALTHY"
}
{
  "InstanceId": "i-0bbdc09f9a55c37de",
  "HealthStatus": "HEALTHY"
}
```

新しく作成されたインスタンス

参考：次のコマンドでAutoScalingのactivityを確認できます。 `$ aws autoscaling describe-scaling-activities`

他に試したこと(参考):

HealthCheckType をELBに変更し、一つのインスタンスはをstop後start
-502 Bad Gatewayエラーが表示され、正常に表示されるまで時間が少し掛かることを分かりました。HealthCheckTypeがEC2なっている場合は502エラーが出ずにELB経由でテストページが正常に表示出来ました。

```
aws autoscaling describe-auto-scaling-groups
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "ELBStack0119-WebServerGroup-1ITTZ6D129R3L",
            "AutoScalingGroupARN": "arn:aws:autoscaling:ap-northeast-1:115757856oupName/ELBStack0119-WebServerGroup-1ITTZ6D129R3L",
            "LaunchConfigurationName": "ELBStack0119-LaunchConfig-1X98XPEQA0LRM"
            "MinSize": 2,
            "MaxSize": 2,
            "DesiredCapacity": 2,
            "DefaultCooldown": 300,
            "AvailabilityZones": [
                "ap-northeast-1a",
                "ap-northeast-1c"
            ],
            "LoadBalancerNames": [],
            "TargetGroupARNs": [
                "arn:aws:elasticloadbalancing:ap-northeast-1:115757856347:target
            ],
            "HealthCheckType": "ELB"
```

```
$ curl ELBSt-Appli-7Q2M2W6ZVQ9U-1141534294.ap-northe
<html>
<head><title>502 Bad Gateway</title></head>
<body bgcolor="white">
<center><h1>502 Bad Gateway</h1></center>
</body>
</html>
```

TEST 2. 障害原因の調査及び回避対策

【概要】
ELB → EC2（AutoScaling使用）→ RDSの構成で、突然ページが表示できなくなりました。
(現在のところ、エラーレスポンスは不明)

1.考えられる障害原因

・ELB のscale out等によるELB IPの変更で、EC2(ウェブサーバー)のApache HTTPDの応答遅延が発生する場合

サーバー台数を増やしてシステムの処理能力を高めるのはScale Outと言います。
＊参考:サーバーのCPU等のスペックを高めるのはScale Up
Scale Outをすると(EC2インスタンスの増加)ELBのIPが変わるので、
その場合EC2(Apache HTTPDの場合)側からの応答遅延が発生する可能性があります。

【Apacheの仕様による応答遅延】
Apache HTTPD起動の時に自分が使う**IPを収集**します。
最初、正しい**IPをELBから**もらい、ちゃんと応答しますが、**ELBのIP変わると**最初もらったIPとことなるので
アクセス失敗になります。
Apache HTTPDには**3回アクセス失敗した時に初期化する機能**がありますので、**最初もらったIPを連動の対象から除外**し,変わたELBのIPを再び収集します。その結果、正常に戻るまで時間がかかります。
そのタイミングで接続すると障害が発生し、ページ表示出来なくなる可能性があります。

・参考:ELBのIP確認は以下のコマンドで確認できます。

```
$ aws ec2 describe-network-interfaces --filters "Name=description,Values=ELB app/ELBSt-Appli-7Q2M2W6ZVQ9U/ba5e5816edcbfe52" | jq -r
'.NetworkInterfaces[] | .Association.PublicIp' | sort
52.199.6.222
54.248.53.71
```

TEST 2. 障害原因の調査及び回避対策

1.考えられる障害原因

・ELBの設定でヘルスチェックタイプがELBに設定され、インスタンスがInService以外になった場合

AutoScalingの**ヘルスチェックタイプ**が**ELBタイプ**になっている場合、
再起動された**インスタンス**がAutoScalingヘルスチェックによりNGと判断され、そのインスタンスは
TerminateされてELBから外されます。 AutoScalingから新しいインスタンスが作られ、ELBが正常に戻る前に少し時間かかります。
そのタイミングでページに接続するとページが表示出来なくなることがあります。

・ SSL証明による接続問題の可能性

ELB経由でHTTPS接続出来きるようにする時にはセキュリティーの為、**SSL証明**を使います。
その**SSL証明書**の**期間切れ**もしくは**インストールされてない**時にELBへの**接続が出来なくなり**ページ表示が出来なくなる可能性も考えられます。

・インスタンスの最大許容接続制限の問題

インスタンスで実行中のapache web serverがある場合、**httpd.conf** にある**MaxClient、ServerLimitの**非効率な設定で、
メモリ負荷になり、応答が出来なくなることがあります。その時にページ表示が出来なくなる問題が発生する可能性があります。

・参考:以下のようにhttpd.confの設定を確認できます。

```
/etc/httpd/conf
[root@ip-10-1-10-21 conf]# vi httpd.conf

# prefork MPM
# StartServers: number of server processes to
start
# MinSpareServers: minimum number of server
processes which are kept spare
# MaxSpareServers: maximum number of server
processes which are kept spare
# ServerLimit: maximum value for MaxClients for
the lifetime of the server
# MaxClients: maximum number of server
processes allowed to start
# MaxRequestsPerChild: maximum number of
requests a server process serves
<IfModule prefork.c>
StartServers       8
MinSpareServers    5
MaxSpareServers   20
ServerLimit      256
MaxClients       256
MaxRequestsPerChild  4000
</IfModule>
```

参考:

・MaxClient:同時接続数、最大子プロセスの数になります。
-デフォルトは256です。
-Topコマンドを叩いてRES(物理メモリ使用率)を確認し、
サーバー物理メモリ量がhttpdプロセスのメモリ使用率 X MaxClient
＋他のプロセスのメモリ使用率より**大きくなる**ように**MaxClientを設定**します。

・ServerLimit:設定可能なサーバープロセス数
-MaxClientが256以上設定しない場合ServerLimitも設定する必要があります。
・StartServers:Apache起動の時の子プロセスの数
・MinspareServers:待機する子プロセスの数
-子プロセスの数がこの設定値より少なくなった場合、その値まで子プロセスの数を増加させます。MaxSpareServerはその反対です。
・MaxRequestPerChild:一個の子プロセスが処理するリクエストの数です。

TEST 2.障害原因の調査及び回避対策

1.考えられる障害原因

・パブリックサブネットのルートテーブルの設定ミスの可能性

ルートテーブルの0.0.0.0/0の宛先がIGWになってない場合、外部からELB経由でインスタンスに接続出来なくなります。

・セキュリティーグループ設定の問題

セキュリティグループはインスタンスの仮想ファイアウォールとして機能をするので設定ミスをするとちゃんと通らないこともあります。
間違ってセキュリティグループを変更するとELB経由でEC2に行けなくなり、ページが表示できなくなる可能性もあります。

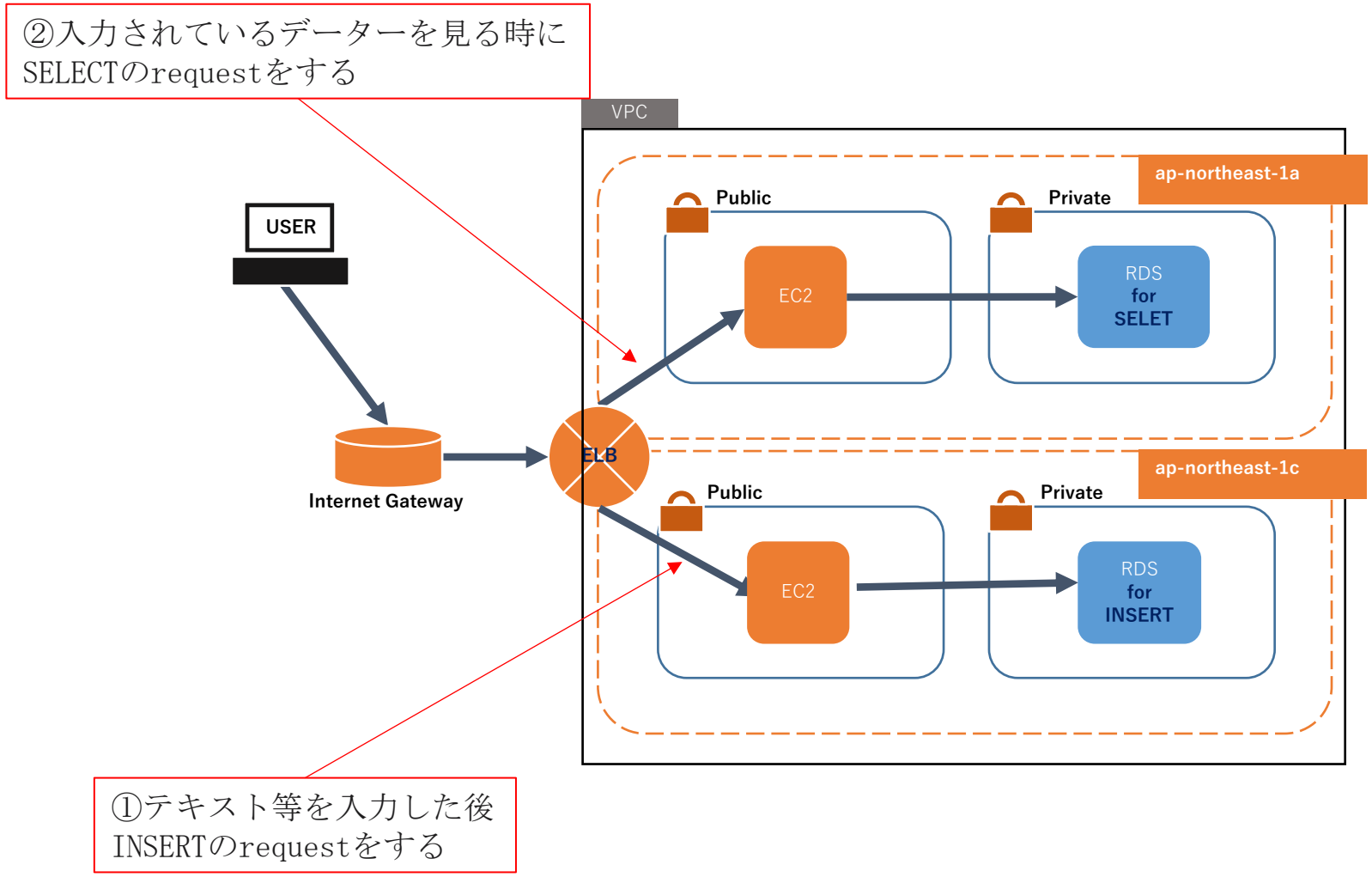
・参考:セキュリティグループにルールを追加する場合は以下のコマンド出来ます。

```
aws ec2 authorize-security-group-ingress --group-name GroupName1 --protocol tcp --port 22 --cidr 123.456.789.012/32
```

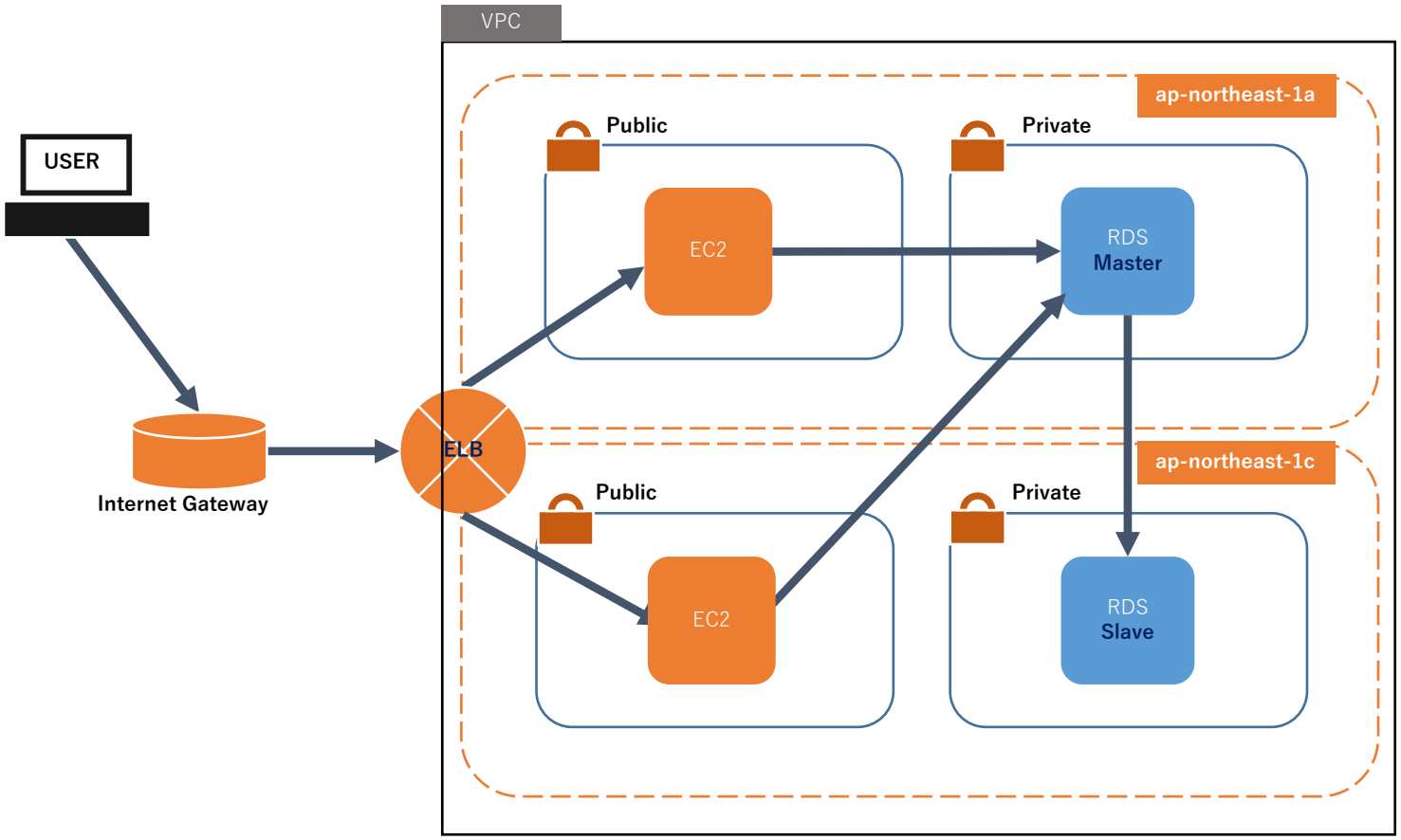
・間違ったELB構成によるトラブル

例:ユーザーのINSERTリクエストをELBが1CにあるDBにSAVEするようにして、次にユーザーがSELECTリクエストするとELBが1AのDBからそのリクエスト応答するような構成をした場合です。
当たり前だとおもいますが、共通でに使えるRESOURCEがあり時に以下のように構成しないように気を付けないと問題が発生します。(RDS AZ構成をする時に勘違いして構築しないように注意する必要があります。)

・間違ったELB構成の例）



・正しいELB構成の例）


VPC
ap-northeast-1a
Public
Private
USER
EC2
RDS
Master
ELB
Internet Gateway
ap-northeast-1c
Public
Private
EC2
RDS
Slave

TEST 2. 障害原因の調査及び回避対策

2.どこを確認してどういう状態ならその箇所が原因であるといえそうなのか？

まずは以下のようにローカルからaws cliコマンドでELBの状況を確認します。（コンソールでも確認可能です。次のページをご参考下さい。）

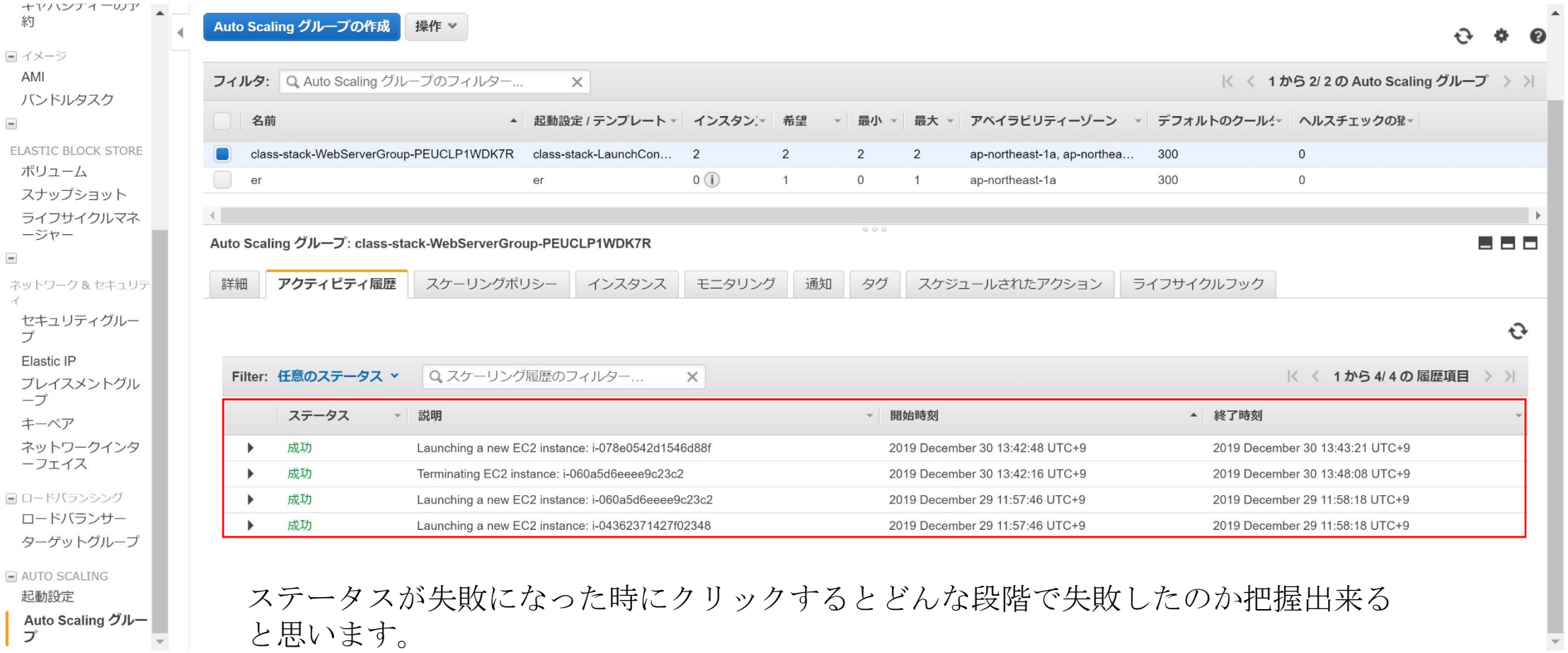
```
$ aws elbv2 describe-load-balancers
{
    "LoadBalancers": [
        {
            "LoadBalancerArn": "arn:aws:elasticloadbalancing:ap-northeast-1:115757856347:loadbalancer/app/class-Appli-P1AJBARDIPT4/2dfa90911ebe89fd",
            "DNSName": "class-Appli-P1AJBARDIPT4-1664726406.ap-northeast-1.elb.amazonaws.com",
            "CanonicalHostedZoneId": "Z14GRHDCWA56QT",
            "CreatedTime": "2019-12-29T02:57:30.240Z",
            "LoadBalancerName": "class-Appli-P1AJBARDIPT4",
            "Scheme": "internet-facing",
            "VpcId": "vpc-0c7d08c78089ea78a",
            "State": {
                "Code": "active"
            },
            "Type": "application",
            "AvailabilityZones": [
                {
                    "ZoneName": "ap-northeast-1c",
                    "SubnetId": "subnet-0640337f58b3338f3",
                    "LoadBalancerAddresses": []
                },
                {
                    "ZoneName": "ap-northeast-1a",
                    "SubnetId": "subnet-09a4c5c825e429262",
                    "LoadBalancerAddresses": []
                }
            ],
            "SecurityGroups": [
                "sg-043307d255bbff4c1"
            ],
            "IpAddressType": "ipv4"
        }
    ]
}
```

ELBが正常に動いているのかを確認します。

TEST 2. 障害原因の調査及び回避対策

2. どこを確認してどういう状態ならその箇所が原因であるといえそうなのか？

コンソールから確認はAuto Scaling Group→Activity Historyでstatusを確認します。



ステータスが失敗になった時にクリックするとどんな段階で失敗したのか把握出来ると思います。

TEST 2. 障害原因の調査及び回避対策

2.どこを確認してどういう状態ならその箇所が原因であるといえそうなのか？

AutoScalingのStatusCodeを確認し問題がないか、もしあったら原因は何なのか確を以下のように確認出来ます。

```
$ aws autoscaling describe-scaling-activities
{
    "Activities": [
        {
            "ActivityId": "3055bfcd-ae3d-daba-6d81-83fd2658d73a",
            "AutoScalingGroupName": "ELBStack0119-WebServerGroup-1ITTZ6D129R3L",
            "Description": "Launching a new EC2 instance: i-0716e93405c80cf7a",
            "Cause": "At 2020-01-29T22:28:28Z an instance was started in respons
e to a difference between desired and actual capacity, increasing the capacity f
rom 1 to 2.",
            "StartTime": "2020-01-29T22:28:30.198Z"
            "EndTime": "2020-01-29T22:29:03Z",
            "StatusCode": "Successful",
            "Progress": 100,
            "Details": "{¥"Subnet ID¥":¥"subnet-0120b27868913fc65¥",¥"Availabili
ty Zone¥":¥"ap-northeast-1a¥"}"
        },
諸略～
```

問題の原因はCauseで把握出来ます。

Statusで正常の有無を確認します。

TEST 2. 障害原因の調査及び回避対策

2.どこを確認してどういう状態ならその箇所が原因であるといえそうなのか？

AWS CLIでロードバランサーとインスタンスのステータス確認する時にDebugオプションを付けるとどんな段階でエラーが出たのか確認出来ます。

```
$ aws elbv2 describe-load-balancers --debug
2020-01-02 17:27:27,817 - MainThread - awscli.clidriver - DEBUG - CLI version: aws-cli/1.16.309 Python/3.7.0 Linux/3.10.0-957.12.2.el7.x86_64 botocore/1.13.45
2020-01-02 17:27:27,818 - MainThread - awscli.clidriver - DEBUG - Arguments entered to CLI: ['elbv2', 'describe-load-balancers', '--debug']
2020-01-02 17:27:27,818 - MainThread - botocore.hooks - DEBUG - Event session-initialized: calling handler <function add_scalar_parsers at 0x7f6cbe7e4510>
2020-01-02 17:27:27,818 - MainThread - botocore.hooks - DEBUG - Event session-initialized: calling handler <function register_uri_param_handler at 0x7f6cbf2fcae8>
2020-01-02 17:27:27,818 - MainThread - botocore.hooks - DEBUG - Event session-initialized: calling handler <function inject_assume_role_provider_cache at 0x7f6cbf2d7bf8>
2020-01-02 17:27:27,821 - MainThread - botocore.hooks - DEBUG - Event session-initialized: calling handler <function attach_history_handler at 0x7f6cc0
```

```
$ aws ec2 describe-instances ¥
> --instance-ids i-078e0542d1546d88f --debug
2020-01-02 17:44:00,411 - MainThread - awscli.clidriver - DEBUG - CLI version: aws-cli/1.16.309 Python/3.7.0 Linux/3.10.0-957.12.2.el7.x86_64 botocore/1.13.45
2020-01-02 17:44:00,411 - MainThread - awscli.clidriver - DEBUG - Arguments entered to CLI: ['ec2', 'describe-instances', '--instance-ids', 'i-078e0542d1546d88f', '--debug']
2020-01-02 17:44:00,411 - MainThread - botocore.hooks - DEBUG - Event session-initialized: calling handler <function add_scalar_parsers at 0x7f36c9400510>
2020-01-02 17:44:00,412 - MainThread - botocore.hooks - DEBUG - Event session-initialized: calling handler <function register_uri_param_handler at 0x7f36c9f18ae8>
```

TEST 2. 障害原因の調査及び回避対策

2.どこを確認してどういう状態ならその箇所が原因であるといえそうなのか？

他に、Instance,ELBのセキュリティグループの設定が正しくなっているかを確認します。

```
$ aws ec2 describe-security-groups --group-ids sg-043307d255bbff4c1
{
    "SecurityGroups": [
        {
            "Description": "Enable HTTP access on the inbound port",
            "GroupName": "class-stack-ALBSecurityGroup-44VFRK8HCG9B",
            "IpPermissions": [
                {
                    "FromPort": 80,
                    "IpProtocol": "tcp",
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "Ipv6Ranges": [],
                    "PrefixListIds": [],
                    "ToPort": 80,
                    "UserIdGroupPairs": []
                }
            ],
省略…
```

TEST 2. 障害原因の調査及び回避対策

2.どこを確認してどういう状態ならその箇所が原因であるといえそうなのか？

他に、インスタンスのステータス情報は以下のように確認できます。

```
aws ec2 describe-instances --instance-id instance_id
```

```
"StateReason": {
   "Message": "Client.VolumeLimitExceeded: Volume limit
exceeded",
  "Code": "Server.InternalError"
},
```

・参考:インスタンスの情報をscreenshotは以下のコマンドで作成出来ます。

```
aws ec2 get-console-screenshot instance_id --instance-id --query 'ImageData' --output text | base64 -d > ./screenShot.jpg
```

DBの状況は以下のコマンドで確認出来ます。

```
aws rds describe-db-instances --db-instance-identifier DB 識別子
```

・参考:DBストレージの問題があった場合以下のコマンドで解決出来ます。

```
aws rds modify-db-instance ¥
  --db-instance-identifier DB 識別子 ¥
  --allocated-storage 60 ¥
  --apply-immediately
```

TEST 2. 障害原因の調査及び回避対策

2.どこを確認してどういう状態ならその箇所が原因であるといえそうなのか？

Linux インスタンスで Apache を実行している場合は、
sudo service httpd status コマンドを叩いて、Apache が実行中であることを確認します。。

```
[ec2-user@ip-10-1-10-83 ~]$ sudo service httpd status
httpd (pid  3024) is running...
```

正常に動いている(running)のかを確認します。Stop場合apacheサーバーが問題がある場合がありますので
アクセスログを確認し原因を調査します。(トラブルシューティング)

インスタンスからELBへの接続テストをする為(設定したポート80で通るか確認)、以下のようにコマンドで確認できます。

```
[ec2-user@ip-10-1-10-83 ~]$ nc -v class-Appli-P1AJBARDIPT4-1664726406.ap-northeast-1.elb.amazonaws.com 80
Connection to class-Appli-P1AJBARDIPT4-1664726406.ap-northeast-1.elb.amazonaws.com 80 port [tcp/http] succeeded!
```

接続テストの結果がsucceededになっていることを確認します。

上記のテスト等でトラブルがあったら、サーバー内のアクセスログの中身を開いて、問題の原因を調べます。

```
[root@ip-10-1-10-21 httpd]# vi  access_log
10.1.20.105 - - [29/Jan/2020:22:29:42 +0000] "GET / HTTP/1.1" 200 46 "-"
"ELB-HealthChecker/2.0"
```

TEST 2. 障害原因の調査及び回避対策

3.障害をAWSで回避または影響を小さくするためにはどうすればいいのか？

・環境構築する時にAutoScaling設定や通信設定が適切なのかを確認します。
→ヘルスチェックのタイプ等の設定を確認。
→SSL証明書の期間を確認(割り当てられなかったら)、ポート設定(80, 443等)が正しいのかを確認。

・環境構築をする時に「dry run」や「debug」optionを使って事前に障害を見つけて対応したほうが良いかと思います。

・環境構築後cloudwatchからアラーム定期的にもらうようにします。
次のページにあるようにインスタンスの状態を定期的にメールに送信するようにすると障害を事前に回避出来る一つの手段として使えると思います。

・参考:次のページにCloudWatch metricsの情報をメールでもらる設定(aws cliで)を一つの例として載せました。

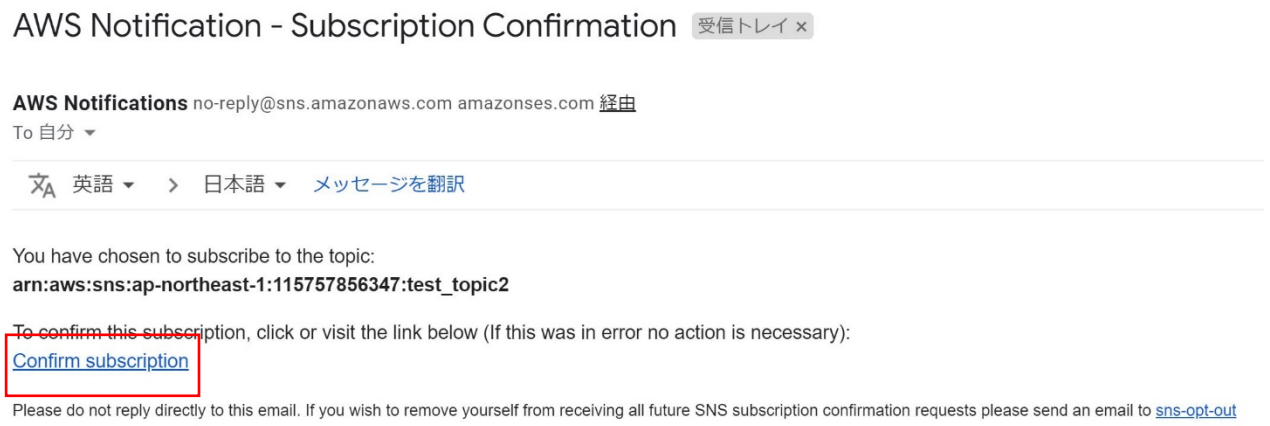
TEST 2. 障害原因の調査及び回避対策

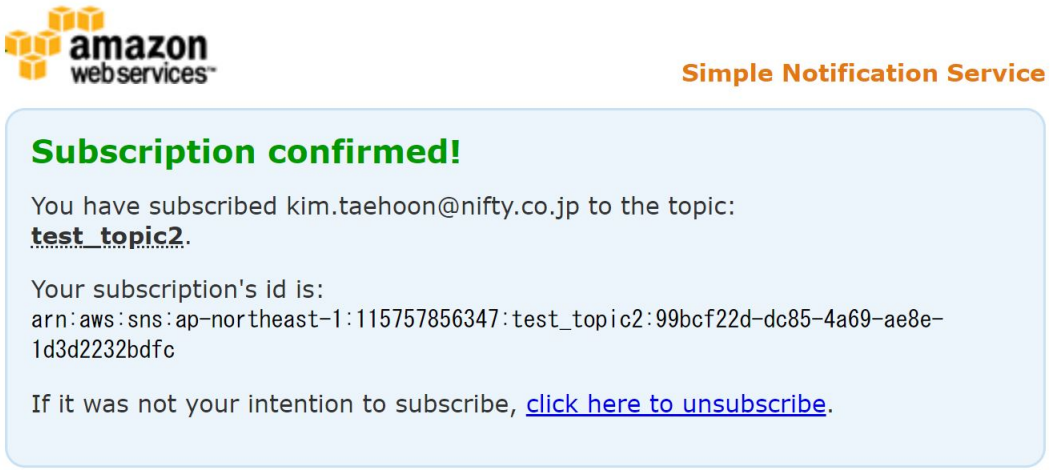
以下のようにAWS CLIでメールを送信を設定します。

```
$ aws sns create-topic --name test_topic2
{
   "TopicArn": "arn:aws:sns:ap-northeast-1:1157578xxxxx:test_topic2“
}
```

```
$ aws sns subscribe --topic-arn arn:aws:sns:ap-northeast-1:1157578xxxxx:test_topic2 ¥
> --protocol email ¥
➢   --notification-endpoint メールアドレス

{
   "SubscriptionArn": "pending confirmation"
}
```

Confirmメールが来るので、confirm subcriptionをクリックするとメール設定完了です。

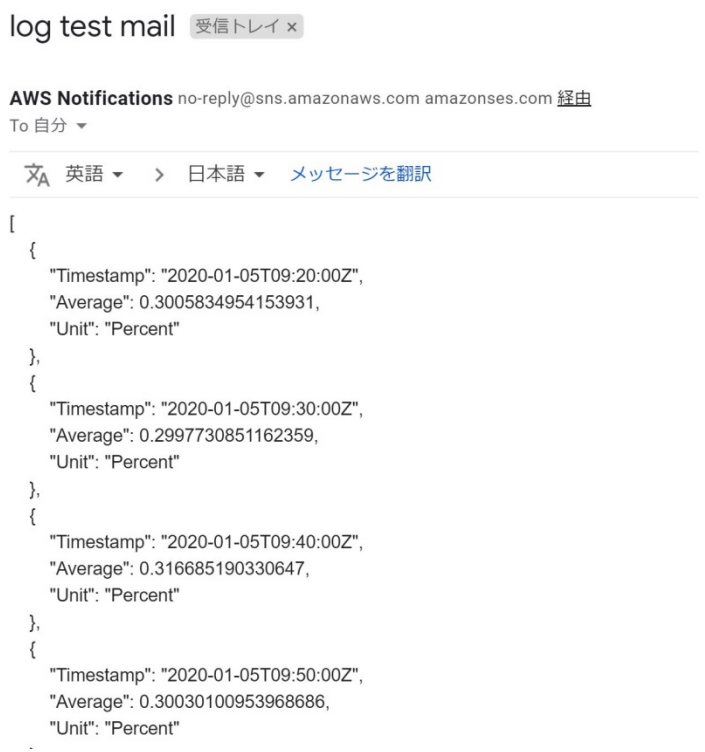




TEST 2. 障害原因の調査及び回避対策

以下のようにmetrics情報を得るスクリプトを作成して、その結果をSNSを使ってメール送信するようにするとメールでもインスタンスの色んなステータスを確認できます。cron 設定をすると定期的にメールでそのインスタンス情報を見ることが出来ます。

```
#!/bin/bash
aws cloudwatch get-metric-statistics --metric-name CPUUtilization --start-time 2020-01-15T00:00:00Z --end-time 2020-01-16T13:00:00Z --per 600 --namespace AWS/EC2
 --statistics Average --dimensions Name=InstanceId,Value=i-07f9162916d000129 --query "sort_by(Datapoints,&Timestamp)[*]" >>log.txt

test=$(cat log.txt)
aws sns publish --topic-arn arn:aws:sns:ap-northeast-1:115757856347:test_topic --subject "log test mail" --message "$test"
```

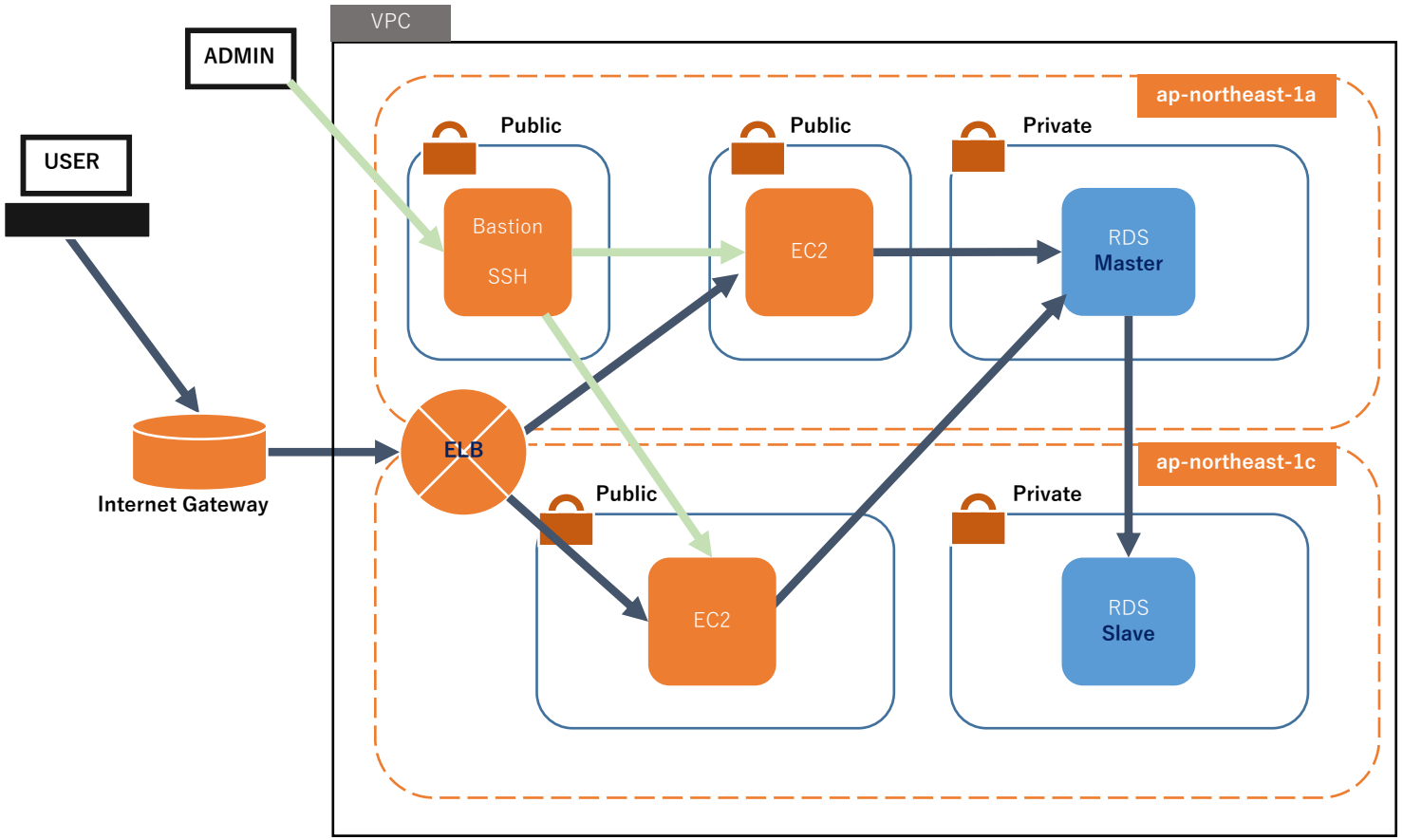


TEST 2. 障害原因の調査及び回避対策

3.その障害をAWSで回避または影響を小さくするためにはどうすればいいのか？

最後に…

・Bastion(踏み台)サーバーを立てて、障害がが発生したら踏み台サーバを経由でメンテナンスをおこない、メンテナンス中でもサービスを継続させます。
踏み台サーバ経由でメンテナンスすることで、障害影響をある程減らすことが出来ると思います。
踏み台サーバ以外も必要に応じて（private serverのex）yum updateなど）NATサーバーやNATゲートウェイも用意した方がいいと思います。



□ その他(参考)

・Cloud Formation template作成

・AWS CLIの設定

VPC 作成-ClassFormation template

VPCは下記の表のようになります。

アベイラビリティーゾーン			
ap-northeast-1a		ap-northeast-1c	
Name	IPv4 CIDR	Name	IPv4 CIDR
test-public-a	10.1.10.0/24	test-public-c	10.1.20.0/24
test-private-a	10.1.30.0/24	test-private-c	10.1.40.0/24

テンプレート 作成手順

設定ファイル(yml)の作成します。

テンプレートの以下のようになります。

①設定の宣言

```
AWSTemplateFormatVersion: "2010-09-09"
Description:
  VPC and Subnet Create
```

テンプレート 作成手順

パラメータを以下のように作成。

```
Metadata:
 "AWS::CloudFormation::Interface":
  ParameterGroups:
   - Label:
      default: "VPC Name "
     Parameters:
      - VPCName
   - Label:
      default: "Create VPC, Subnet"
     Parameters:
      - VPCCIDR
      - PublicSubnetACIDR
      - PublicSubnetCCIDR
      - PrivateSubnetACIDR
      - PrivateSubnetCCIDR
  ParameterLabels:
   VPCCIDR:
    default: "VPC CIDR Block"
   PublicSubnetACIDR:
    default: "PublicSubnetA CIDR Block"
   PublicSubnetCCIDR:
    default: "PublicSubnetC CIDR Block"
   PrivateSubnetACIDR:
    default: "PrivateSubnetA CIDR Blcok"
   PrivateSubnetCCIDR:
    default: "PrivateSubnetC CIDR Block"
```

パラメータのデフォルト値を作成。

```
Parameters:
 VPCName:
  Type: String

 VPCCIDR:
  Type: String
  Default: "10.1.0.0/16"

 PublicSubnetACIDR:
  Type: String
  Default: "10.1.10.0/24"

 PublicSubnetCCIDR:
  Type: String
  Default: "10.1.20.0/24"

 PrivateSubnetACIDR:
  Type: String
  Default: "10.1.30.0/24"

 PrivateSubnetCCIDR:
  Type: String
  Default: "10.1.40.0/24"

Resources:
```

VPC、インターネットゲートウェイ作成及びattach設定

```
VPC:
  Type: "AWS::EC2::VPC"
  Properties:
   CidrBlock: !Ref VPCCIDR
   EnableDnsSupport: "true"
   EnableDnsHostnames: "true"
   InstanceTenancy: default
   Tags:
    - Key: Name
      Value: !Sub "${VPCName}-vpc"

# InternetGateway Create
 InternetGateway:
  Type: "AWS::EC2::InternetGateway"
  Properties:
   Tags:
    - Key: Name
      Value: !Sub "${VPCName}-igw"

# IGW Attach
 InternetGatewayAttachment:
  Type: "AWS::EC2::VPCGatewayAttachment"
  Properties:
   InternetGatewayId: !Ref InternetGateway
   VpcId: !Ref VPC
```

テンプレート 作成手順

②Subnet作成設定

```
  PublicSubnetA:
    Type: "AWS::EC2::Subnet"
    Properties:
      AvailabilityZone: "ap-northeast-1a"
      CidrBlock: !Ref PublicSubnetACIDR
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub "${VPCName}-public-subnet-a"

# Create Public SubnetC
  PublicSubnetC:
    Type: "AWS::EC2::Subnet"
    Properties:
      AvailabilityZone: "ap-northeast-1c"
      CidrBlock: !Ref PublicSubnetCCIDR
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub "${VPCName}-public-subnet-c"

# Private SubnetA Create
  PrivateSubnetA:
    Type: "AWS::EC2::Subnet"
    Properties:
      AvailabilityZone: "ap-northeast-1a"
      CidrBlock: !Ref PrivateSubnetACIDR
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub "${VPCName}-private-subnet-a"

# Create Private SubnetC
  PrivateSubnetC:
    Type: "AWS::EC2::Subnet"
    Properties:
      AvailabilityZone: "ap-northeast-1c"
      CidrBlock: !Ref PrivateSubnetCCIDR
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub "${VPCName}-private-subnet-c"
```

③RouteTable作成設定

```
PublicRouteTableA:
    Type: "AWS::EC2::RouteTable"
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub "${VPCName}-public-route-a"

# Create Public RouteTableC
  PublicRouteTableC:
    Type: "AWS::EC2::RouteTable"
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub "${VPCName}-public-route-c"

# Create Private RouteTableA
  PrivateRouteTableA:
    Type: "AWS::EC2::RouteTable"
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub "${VPCName}-private-route-a"

# Create Private RouteTableC
  PrivateRouteTableC:
    Type: "AWS::EC2::RouteTable"
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub "${VPCName}-private-route-c"
```

④Routingの設定

```
PublicRouteA:
    Type: "AWS::EC2::Route"
    Properties:
      RouteTableId: !Ref PublicRouteTableA
      DestinationCidrBlock: "0.0.0.0/0"
      GatewayId: !Ref InternetGateway

# PublicRouteC Create
  PublicRouteC:
    Type: "AWS::EC2::Route"
    Properties:
      RouteTableId: !Ref PublicRouteTableC
      DestinationCidrBlock: "0.0.0.0/0"
      GatewayId: !Ref InternetGateway

# ------------------------------------------------------------#
# RouteTable Associate
# ------------------------------------------------------------#
# PublicRouteTable Associate SubnetA
  PublicSubnetARouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PublicSubnetA
      RouteTableId: !Ref PublicRouteTableA

# PublicRouteTable Associate SubnetC
  PublicSubnetCRouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PublicSubnetC
      RouteTableId: !Ref PublicRouteTableC

# PrivateRouteTable Associate SubnetA
  PrivateSubnetARouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PrivateSubnetA
      RouteTableId: !Ref PrivateRouteTableA

# PrivateRouteTable Associate SubnetC
  PrivateSubnetCRouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PrivateSubnetC
      RouteTableId: !Ref PrivateRouteTableC
```

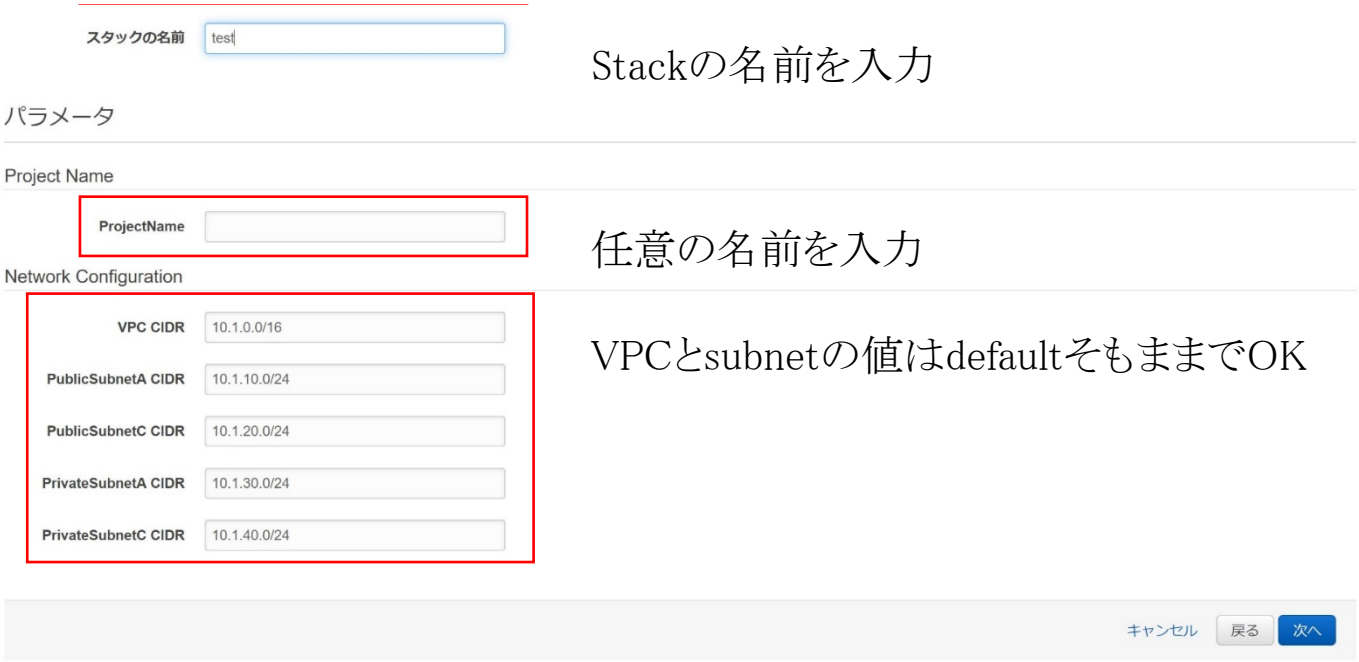
⑥出力

```
Outputs:
# VPC
  VPC:
    Value: !Ref VPC
    Export:
      Name: !Sub "${VPCName}-vpc"

  VPCCIDR:
    Value: !Ref VPCCIDR
    Export:
      Name: !Sub "${VPCName}-vpc-cidr"

# Subnet
  PublicSubnetA:
    Value: !Ref PublicSubnetA
    Export:
      Name: !Sub "${VPCName}-public-subnet-a"

  PublicSubnetACIDR:
    Value: !Ref PublicSubnetACIDR
    Export:
      Name: !Sub "${VPCName}-public-subnet-a-cidr"

  PublicSubnetC:
    Value: !Ref PublicSubnetC
    Export:
      Name: !Sub "${VPCName}-public-subnet-c"

  PublicSubnetCCIDR:
    Value: !Ref PublicSubnetCCIDR
    Export:
      Name: !Sub "${VPCName}-public-subnet-c-cidr"

  PrivateSubnetA:
    Value: !Ref PrivateSubnetA
    Export:
      Name: !Sub "${VPCName}-private-subnet-a"
```

```
  PrivateSubnetACIDR:
    Value: !Ref PrivateSubnetACIDR
    Export:
      Name: !Sub "${VPCName}-private-subnet-a-cidr"

  PrivateSubnetC:
    Value: !Ref PrivateSubnetC
    Export:
      Name: !Sub "${VPCName}-private-subnet-c"

  PrivateSubnetCCIDR:
    Value: !Ref PrivateSubnetCCIDR
    Export:
      Name: !Sub "${VPCName}-private-subnet-c-cidr"

# Route
  PublicRouteTableA:
    Value: !Ref PublicRouteTableA
    Export:
      Name: !Sub "${VPCName}-public-route-a"

  PublicRouteTableC:
    Value: !Ref PublicRouteTableC
    Export:
      Name: !Sub "${VPCName}-public-route-c"

  PrivateRouteTableA:
    Value: !Ref PrivateRouteTableA
    Export:
      Name: !Sub "${VPCName}-private-route-a"

  PrivateRouteTableC:
    Value: !Ref PrivateRouteTableC
    Export:
      Name: !Sub "${VPCName}-private-route-c"
```

テンプレート 作成手順

CloudFormationで先程作成したVPC設定ymlファイル(テンプレート)を使ってVPC構築



Stackの名前を入力

任意の名前を入力

VPCとsubnetの値はdefaultそもままでOK

問題なく完了するとCREATE_COMPLETE表示されます。

環境構築(本番)

以下のテンプレートを使って環境構築を進めます。

https://s3-ap-northeast-1.amazonaws.com/cm-ope-exam-prod/CM-Ope-CFnTemplates.yaml?versionId=null

URLの入れます

環境構築(本番)

以下のように値を入力して環境構築をします。

AWS CLIの設定

下記のURLでAWS Access Key ID とAWS Secret Access Keyを作ります。

https://console.aws.amazon.com/iam/home#/users/kim2?section=security_credentials

上記のURLに入って「Identity and Access Management (IAM)」の設定をします。

1

ユーザー→ユーザーを追加選択

2

名前を入力後次へ

3

ユーザーグループに追加後次へ

AWS CLIの設定

4

EC2FullAccess権限を検索して選択後次へ

5

アクセスキーIDとシークレットキーをメモして次の設定せで使います

AWS CLIの設定

もし,権限に問題があったら、以下のようにユーザーメニューでアクセス権限を追加します 。

AWS CLIの設定

6

ローカル環境でAWS CLIを設定

```
$ aws configure
AWS Access Key ID [****************TNUQ]:
AWS Secret Access Key [****************yKXL]:
Default region name [ap-northeast-1]:
Default output format [json]:
```

IAMでメモしたキーIDとアクセスキーを入力、regionはap-northeast-1にoutput 形式はjsonにします。

instance確認

```
$ aws ec2 describe-tags --query 'Tags[?ResourceType==`instance`] | [?Key==`Name`].{ResourceId:ResourceId,ResourceName:Value}'
--output table
```

動作確認

```
$ aws ec2 start-instances --instance-ids i-0xxxxxxxxxxxxxx
{
    "StartingInstances": [
        {
            "CurrentState": {
                "Code": 16,
                "Name": "running"
            },
            "InstanceId": "i-0xxxxxxxxxxxxxx",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
```

jqコマンドのインストール

```
$ sudo yum install jq
```

インスタンスIDのみ取得

```
$ aws ec2 describe-instances | jq '.Reservations[].Instances[].InstanceId'
"i-0f
"i-04
"i-0c
"i-0c
"i-0b
```

AWS CLIの設定　その他

jqコマンドのインストール

```
$ sudo yum install jq
```

jqコマンドは以下のように活用出来ます。

インスタンスIDのみ取得

```
$ aws ec2 describe-instances | jq '.Reservations[].Instances[].InstanceId'
"i-0f
"i-04
"i-0c
"i-0c
"i-0b
```

インスタンスID、インスタンスタイプ、publicIPを取得

```
$ aws ec2 describe-instances | jq '.Reservations[].Instances[] | {InstanceId, InstanceType, PublicIpAddress}'
```

インスタンスID、インスタンスタイプ、privateIPを取得

```
$ aws ec2 describe-instances | jq '.Reservations[].Instances[] | {InstanceId, InstanceType, PrivateIpAddress}'
```