AWS

Description

In this project we will go through how to setup a test env. with AWS Cloudformation that create a Application LoadBalancer that's connected with a AutoScale Group and policy that scale up when instance is over 80% over 60 sec timespace.

Parameters

With specified paramters the template will be easier for non-auther when needed to change to another availalitityzone or a users specific private and secret key for connect to instances.

```
Parameters:
 KeyParameter:
   Type: "AWS::EC2::KeyPair::KeyName"
   Default: AWStest
 AZparameter:
   Type: List<AWS::EC2::AvailabilityZone::Name>
   Default: eu-west-1a, eu-west-1b, eu-west-1c
 ImageParameter:
   Type: AWS::EC2::Image::Id
   Default: ami-0dab0800aa38826f2
```

Networking

With creating a stack unique VPC for creating a segrigation from other instances that maybe up and running.

VPC

```
myVPC:
  Type: AWS::EC2::VPC
  Properties:
    CidrBlock: 10.0.0.0/16
    EnableDnsSupport: 'true'
    EnableDnsHostnames: 'true'
    Tags:
      - Key: 'Name'
        Value: !Sub '${AWS::StackName}-VPC'
```

Subnet

Create 3 subnets, 1 for each availabilityZone in EU-West-1

```
# # Create Subnet A
  subnetA:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref myVPC
      AvailabilityZone: { "Fn::Select" :[ "0", {"Ref": "AZparameter" }]}
      CidrBlock: 10.0.10.0/24
      MapPublicIpOnLaunch: True
      Tags:
        - Key: 'Name'
          Value: !Sub '${AWS::StackName}-SubnetA'
# # Create Subnet B
  subnetB:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref myVPC
      AvailabilityZone: { "Fn::Select" :[ "1", {"Ref": "AZparameter" }]}
      CidrBlock: 10.0.20.0/24
      MapPublicIpOnLaunch: True
      Tags:
        - Key: 'Name'
          Value: !Sub '${AWS::StackName}-SubnetB'
# # Create Subnet C
  subnetC:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref myVPC
      AvailabilityZone: { "Fn::Select" :[ "2", {"Ref": "AZparameter" }]}
      CidrBlock: 10.0.30.0/24
      MapPublicIpOnLaunch: True
      Tags:
        - Key: 'Name'
          Value: !Sub '${AWS::StackName}-SubnetC'
```

Route

With creating your own VPC its needed to setup both a route and routetable so VPC kan communicate with subnets and also gateway to internet.

```
# Routes
  RouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref myVPC
      Tags:
        - Key: 'Name'
          Value: !Sub '${AWS::StackName}-RouteTable'
  routeName:
    Type: AWS::EC2::Route
    Properties:
```

```
      RouteTableId: !Ref RouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref myInternetGateway
  mySubnetRouteTableAssociationA:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId:
        !Ref subnetA
      RouteTableId:
        Ref: RouteTable
  mySubnetRouteTableAssociationB:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId:
        !Ref subnetB
      RouteTableId:
        Ref: RouteTable
  mySubnetRouteTableAssociationC:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId:
        !Ref subnetC
      RouteTableId:
        Ref: RouteTable
```

Gateway

```
# Gateway
  myInternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: 'Name'
          Value: !Sub '${AWS::StackName}-InternetGateway:'
  AttachGateway:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref myVPC
      InternetGatewayId: !Ref myInternetGateway
```

SecureGroups

With setting up security groups you can control what instances connect with another and out to the internet, here we have set up so the EC2 are only open on port 22 to the external network but port 80 only are accessible through the loadbalancer "ALB"

```
# Create ALB-SecGrp http
  ALBsecGroupNameHTTP:
    Type: AWS::EC2::SecurityGroup
```

```
      Properties:
        GroupName: !Sub '${AWS::StackName}-ALB-SecGrpHTTP'
        GroupDescription: 'AppLoadBalancer - Allow HTTP - Anywhere'
        VpcId: !Ref myVPC
        SecurityGroupIngress:
          - IpProtocol: 'TCP'
            FromPort: 80
            ToPort: 80
            CidrIp: 0.0.0.0/0
        Tags:
          - Key: 'Name'
            Value: !Sub '${AWS::StackName}-ALB-SecGrpHTTP'
  # Create SecGrp http
    secGroupNameHTTP:
      Type: AWS::EC2::SecurityGroup
      Properties:
        GroupName: !Sub '${AWS::StackName}-SecGrpHTTP'
        GroupDescription: 'Allow HTTP - to ALB'
        VpcId: !Ref myVPC
        SecurityGroupIngress:
          - IpProtocol: 'TCP'
            FromPort: 80
            ToPort: 80
            SourceSecurityGroupId: !Ref ALBsecGroupNameHTTP
        Tags:
          - Key: 'Name'
            Value: !Sub '${AWS::StackName}-SecGrpHTTP'
  # Create SecGrp ssh
    secGroupNameSSH:
      Type: AWS::EC2::SecurityGroup
      Properties:
        GroupName: !Sub '${AWS::StackName}-SecGrpSSH'
        GroupDescription: 'Allow SSH - Anywhere'
        VpcId: !Ref myVPC
        SecurityGroupIngress:
          - IpProtocol: 'TCP'
            FromPort: 22
            ToPort: 22
            CidrIp: 0.0.0.0/0
        Tags:
          - Key: 'Name'
            Value: !Sub '${AWS::StackName}-SecGrpSSH'
```

Template

LaunchTemplate

Here we create a launchtemplate with paramter for image and securegroups for the EC2 that will be using for our autoscale group and policy. We also use the UserData switch in AWS::EC2::LaunchTemplate for update OS and install webserver package nginx. We also use command sed for update the index file for our website so it show my name and also with servers webpage you are connected to.

```
# Create Launchtemplate
  launchTemplateName:
    Type: AWS::EC2::LaunchTemplate
    Properties:
      LaunchTemplateName: !Sub '${AWS::StackName}-Test2000'
      LaunchTemplateData:
        KeyName:
          Ref: KeyParameter
        InstanceType: 't2.micro'
        ImageId: !Ref "ImageParameter"
        SecurityGroupIds:
          - !Ref secGroupNameSSH
          - !Ref secGroupNameHTTP
        Monitoring:
          Enabled: true
        UserData: !Base64 |
         #!/bin/bash
         dnf update -y && dnf install nginx -y
         sed -i 's\<h1>Welcome to nginx!</h1>\<h1>Hello World</h1>'
/usr/share/nginx/html/index.html
         sed -i '/<h1>/a My name is Nilsson, Fredrik Nilsson'
/usr/share/nginx/html/index.html
         sed -i -e "/<p><em>/a Connected to server: $(hostname -s)"
/usr/share/nginx/html/index.html
         systemctl start nginx && systemctl enable nginx
```

Auto Scale Group

Here we create our AutoScale group that has a desirecapacity 1 so there will always be atleast 1 EC2 and there is set a maximum limited to 3 EC2 at the same time, it is also using a dynamic scalingpolicy so it scale up 1 instance when the scaling group of CPU value is over 80% for a period of 60 seconds.

```
# Create AutoScaleGroup
  asg:
    Type: AWS::AutoScaling::AutoScalingGroup
    Properties:
      AutoScalingGroupName: !Sub "${AWS::StackName}-ASG"
      MaxSize: '3'
      MinSize: '1'
      DesiredCapacity: '1'
      LaunchTemplate:
        LaunchTemplateId: !Ref launchTemplateName
        Version: !GetAtt launchTemplateName.LatestVersionNumber
      VPCZoneIdentifier:
        - !Ref subnetA
        - !Ref subnetB
        - !Ref subnetC
      MetricsCollection:
        - Granularity: 1Minute
          Metrics:
```

```
            - GroupMinSize
            - GroupMaxSize
      TargetGroupARNs:
        - !Ref MyTargetGroup
      Tags:
        - Key: Name
          Value: !Sub "${AWS::StackName}-ASG"
          PropagateAtLaunch: false
# Assign ScalingPolicy
  MyScaleUpPolicy:
    Type: AWS::AutoScaling::ScalingPolicy
    Properties:
      PolicyType: TargetTrackingScaling
      AutoScalingGroupName: !Ref asg
      EstimatedInstanceWarmup: 60
      TargetTrackingConfiguration:
        PredefinedMetricSpecification:
          PredefinedMetricType: ASGAverageCPUUtilization
        TargetValue: '80'
        DisableScaleIn: false
      Cooldown: 60  # Adjust the cooldown period as needed (in seconds)
```

AppLoadbalancer

This loadbalancer is internet connected so it can connect to the internet and also with help of our securitygroups and subnets communicate with our EC2 we are creating so we minimize our "footprint" of our webservers to the widewebn so we only alow traffic through port 80 with our loadbalancer.

```
# Create Application Loadbalancer
  ALB:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      IpAddressType: 'ipv4'
      Name: !Sub "${AWS::StackName}-ALB"
      Scheme: internet-facing
      SecurityGroups:
        - !Ref ALBsecGroupNameHTTP
      Subnets:
        - !Ref subnetA
        - !Ref subnetB
        - !Ref subnetC
      Type: 'application'

# Create Target Group
  MyTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      Name: !Sub '${AWS::StackName}-TargetGroup'
      Port: 80
      Protocol: HTTP
      VpcId: !Ref myVPC
```

```
# Port listning
  HTTPlistener:
   Type: 'AWS::ElasticLoadBalancingV2::Listener'
   Properties:
      DefaultActions:
       - Type: forward
         ForwardConfig:
          TargetGroups:
            - TargetGroupArn: !Ref MyTargetGroup
      LoadBalancerArn: !Ref ALB
      Port: 80
      Protocol: 'HTTP'
```

Useful Commands

To start and update a stack we can use following powershell command after install AWSCLI, guide for that can be found here: Link

With the `Deploy` command we can both create and update exsisting completed stack, but can be change for `create` for only create stack.

```
aws cloudformation deploy --template-file ./AWS/uppgift1.yaml --stack-name [paste your unique stackname here]
```

If we what to delete a stack we can use the command.

```
aws cloudformation delete-stack --stack-name [Paste name of stack for deletion]
```

other command that can be useful is to be able to list the availability zones for a specific region, in this example we fetch in us-west-1 region.

```
 aws ec2 describe-availability-zones --region us-west-1 --query 'AvailabilityZones[].ZoneName'
```

GitHub

Link to Github repo: https://github.com/Norra-frenzu/AWS