Deploy a High-Availability Web App using CloudFormation

REVIEW | CODE REVIEW 9 | HISTORY

server.yml 8

```
Description: >
  Narotham Sai / Udacity Project

Parameters:
  EnvironmentName:
    Description: An environment name that will be prefixed to environment resource names
    Type: String

  EC2InstanceAMI:
    Description: EC2 Image AMI ID
    Type: String
    Default: ami-003634241a8fcdec0

  WebAppS3Name:
    Description: Web App S3 Bucket Name
    Type: String
    Default: narotham-udagram



Resources:
  LBSecGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Allow http to load balancer
      SecurityGroupEgress:
        - IpProtocol: tcp
          FromPort: 0
          ToPort: 65535
          CidrIp: 0.0.0.0/0
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
```

```
      VpcId:
        Fn::ImportValue: !Sub "${EnvironmentName}-VPCID"

  WebServerSecGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Allow http to our host and SSH from local only
      SecurityGroupEgress:
        - IpProtocol: tcp
          FromPort: 0
          ToPort: 65535
          CidrIp: 0.0.0.0/0
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
      VpcId:
        Fn::ImportValue: !Sub "${EnvironmentName}-VPCID"
```

AWESOME

Security groups always first :) It is great that you have security groups at the beginning of your code, it helps the readability of your script.

```

  RoleEC2ReadOnlyS3:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - ec2.amazonaws.com
            Action:
              - "sts:AssumeRole"
      ManagedPolicyArns:
        - "arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
      Path: "/"
```

SUGGESTION

Awesome practice of defining instance role! As a suggestion, you can also define policy for the role here, for instance you can define the actions that ReadOnly policy assigns to the role. This helps the readability of your code.

```

  EC2InstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Path: "/"
      Roles:
        - !Ref RoleEC2ReadOnlyS3

  WebAppLaunchConfig:
    Type: AWS::AutoScaling::LaunchConfiguration
    Properties:
      UserData:
        Fn::Base64: !Sub |
```

```
          #!/bin/bash
          snap install aws-cli --classic
          apt-get update -y
          apt-get install unzip apache2 -y
          systemctl start apache2.service
          systemctl enable apache2.service
          cd /var/www/html
          rm index.html
          aws s3 cp s3://${WebAppS3Name}/udagram.zip .
          unzip -o udagram.zip
      ImageId: !Ref EC2InstanceAMI
      SecurityGroups:
        - Ref: WebServerSecGroup
      InstanceType: t3.medium
      IamInstanceProfile: !Ref EC2InstanceProfile
      BlockDeviceMappings:
        - DeviceName: "/dev/sda1"
          Ebs:
            VolumeSize: "10"
```

AWESOME

Great selection of instance type. It is a good practice that you don't have a key for private subnet instance. You can also have instance type and volume size as parameters at the top of your code.

```

  WebAppGroup:
    Type: AWS::AutoScaling::AutoScalingGroup
    Properties:
      VPCZoneIdentifier:
        - Fn::ImportValue: !Sub "${EnvironmentName}-PRIV-NETS"
```

AWESOME

Auto scaling group in private subnets :) nice practice.

```
      LaunchConfigurationName:
        Ref: WebAppLaunchConfig
      MinSize: "4"
      MaxSize: "6"
      TargetGroupARNs:
        - Ref: WebAppTargetGroup

  WebAppLB:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Subnets:
        - Fn::ImportValue: !Sub "${EnvironmentName}-PUB1-SN"
        - Fn::ImportValue: !Sub "${EnvironmentName}-PUB2-SN"
```

AWESOME

Great design for availability. Nice practice of deploying Load balancer on public subnets :)

```
      SecurityGroups:
        - Ref: LBSecGroup

  Listener:
```

```
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
        - Type: forward
          TargetGroupArn:
            Ref: WebAppTargetGroup
      LoadBalancerArn:
        Ref: WebAppLB
      Port: "80"
```

AWESOME

Great, port 80 for load balancer listener.

```
      Protocol: HTTP

  ALBListenerRule:
    Type: AWS::ElasticLoadBalancingV2::ListenerRule
    Properties:
      Actions:
        - Type: forward
          TargetGroupArn: !Ref "WebAppTargetGroup"
      Conditions:
        - Field: path-pattern
          Values: [/]
      ListenerArn: !Ref "Listener"
      Priority: 1

  WebAppTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      HealthCheckIntervalSeconds: 30
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 25
      HealthyThresholdCount: 2
      Port: 80
      Protocol: HTTP
```

AWESOME

Awesome, target group communication through http.

```
      UnhealthyThresholdCount: 5
      VpcId:
        Fn::ImportValue:
          Fn::Sub: "${EnvironmentName}-VPCID"

Outputs:
  LBDNS:
    Description: URl of the Load Balancer
    Value:
      !Join
      - ''
      - - 'http://'
        - !GetAtt WebAppLB.DNSName
```

AWESOME

It is very great that you have output for LB URL, improves readability of your code significantly.

```
172     Export:
173       Name: !Sub ${EnvironmentName}-LBDNS
174
```

▸ **README.md** 1

▸ **network.yml**

RETURN TO PATH