Servers & Security Groups

July 6, 2020

1 Security Groups

The following is the syntax required to create a `SecurityGroup`:

Security groups are assoicated with specific resources and not tied to the subnets

```
Type: AWS::EC2::SecurityGroup
Properties:
  GroupDescription: String
  GroupName: String
  SecurityGroupEgress:
    - Egress
  SecurityGroupIngress:
    - Ingress
  Tags:
    - Tag
  VpcId: String
```

Although they are not required, the `SecurityGroupEgress` and `SecurityGroupIngress` property rules are the most critical to the `SecurityGroup` as it defines where the traffic will go. While `SecurityGroupEgress` defines outbound traffic, `SecurityGroupIngress` defines the inbound traffic

1.1 Ingress Rules and Egress Rules

- Ingress rules are for inbound traffic, and egress rules are for the outbound traffic
- Ingress rules restrict or allow traffic trying to reach our resources on specific ports
- Egress rules restrict or allow traffic originating from our server –typically we are ok allowing all outbound traffic without restrictions as this doesn't pose a risk for a security breach

1.2 Traffic is blocked by default

- In cloud traffic is completely blocked, so you have to explicitly open ports to allow traffic in and out. This is a general networking concept

1.3 Limit inbound traffic for security

- For ingress rules, we want to limit inbound traffic, for security, to a single port or just a handful of ports required by the application we are running
- If it's a public web server, for example, it will require `port 80` open to the world (World = `0.0.0.0/0`)

- Should you need the SSH port open, restrict this port only to specify IP adress
- should you need the SSH port open, restrict this port only to your specific IP adress

1.4 For outbound traffic, give full access

For egress rules, we want to give this resource full access to the internet so we give egress access to all ports, from `0` all the way to `65535`

2 Creating AutoScaling Group

The autoscaling group must be given a launch configuration and a scaling policy

The following is the syntax used for `AutoScaling` `LaunchConfiguration`:

```
Type: AWS::AutoScaling::LaunchConfiguration
Properties:
  AssociatePublicIpAddress: Boolean
  BlockDeviceMappings:
    - BlockDeviceMapping
  ClassicLinkVPCId: String
  ClassicLinkVPCSecurityGroups:
    - String
  EbsOptimized: Boolean
  IamInstanceProfile: String
  ImageId: String
  InstanceId: String
  InstanceMonitoring: Boolean
  InstanceType: String
  KernelId: String
  KeyName: String
  LaunchConfigurationName: String
  PlacementTenancy: String
  RamDiskId: String
  SecurityGroups:
    - String
  SpotPrice: String
  UserData: String
```

The `ImageId` and `Instance Type` are the only required properties for a `LaunchConfiguration`. Howerver, there are many useful properties you will likely want to include

In the example below we have done the following: - Specified 10gbs for our `VolumeSize` - Referenced the previously defined `WebServerSecGroup` for our `SecurityGroup` - Set our `InstanceType` to `t3.medium` for our EC2 Instance

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

```
          #!/bin/bash
          apt-get update -y
          apt-get install unzip awscli -y
          apt-get install apache2 -y
          systemctl start apache2.service
          cd /var/www/html
          aws s3 cp s3://udacity-demo-1/udacity.zip .
          unzip -o udacity.zip
      ImageId: ami-005bdb005fb00e791
      IamInstanceProfile: !Ref ProfileWithRolesForOurApp
      SecurityGroups:
      - Ref: WebServerSecGroup
      InstanceType: t3.small
      BlockDeviceMappings:
      - DeviceName: "/dev/sdk"
        Ebs:
          VolumeSize: '10'
```

The following is the required syntax for `TargetGroup`

```
Type: AWS::ElasticLoadBalancingV2::TargetGroup
Properties:
  HealthCheckEnabled: Boolean
  HealthCheckIntervalSeconds: Integer
  HealthCheckPath: String
  HealthCheckPort: String
  HealthCheckProtocol: String
  HealthCheckTimeoutSeconds: Integer
  HealthyThresholdCount: Integer
  Matcher:
    Matcher
  Name: String
  Port: Integer
  Protocol: String
  Tags:
    - Tag
  TargetGroupAttributes:
    - TargetGroupAttribute
  TargetType: String
  Targets:
    - TargetDescription
  UnhealthyThresholdCount: Integer
  VpcId: String
```

`Health Checks` are the requests your `Application Load Balancer` sends to its registered targets. These periodic requests test the status of these targets. You can see us defining our `Health Check` properties in the example below:

```
 WebAppTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
```

```
Properties:
  HealthCheckIntervalSeconds: 35
  HealthCheckPath: /
  HealthCheckProtocol: HTTP
  HealthCheckTimeoutSeconds: 30
  HealthyThresholdCount: 2
  Port: 80
  Protocol: HTTP
  UnhealthyThresholdCount: 5
  VpcId:
    Fn::ImportValue:
      Fn::Sub: "${EnvironmentName}-VPCID"
```

In the above example we specify the following: - The port where our targets receive traffic - `Port: 80` - The protocol the load balancer uses when performing health checks on targets - `HealthCheckProtocol: HTTP` - The time it takes to determine a non-responsive target is unhealthy- `HealthCheckingIntervalSeconds: 35` - The number of health/unhealthy checks required to change the health status - `HealthyThresholdCount: 2` `UnhealthyThreshholdCount: 5`

3 Security Group Error

For the load balancer, we basically have port `80` as both inbound and outbound

Thus your basically limiting the load balancer to only be able to talk with `HTTP`

That is why we add port `8080` as shown below