COURSEWARE

AWS Auto-Scaling Group CLI

Contents

Overview

Auto-Scaling monitors your applications and adjusts the capacity to ensure a steady performance while keeping costs low.
Depending on the configurations made by the user, you can utilise this feature such that when your servers CPU reaches 80% capacity, your auto-scaling group will create another EC2 instance and redirect the load between the old and new EC2 instance, therefore, reducing the CPU usage.

Tutorial

This tutorial will cover how we can create an autoscaling group using the AWS CLI. We
must create a Security Group, an Elastic Load Balancer and Launch Configuration before we
can create an AutoScaling Group.

We are going to first create a security group for our launch configuration to be assigned to. Without
this, the launch configuration will have no rules set, so we wont be able to SSH into the
instance.

Creating a Security Group

1. We will be creating a security group without any rules attached initially, as the CLI
   does not include any flags to add rules together.

Run the following to create a security group called QASAMPLE2020:

```
aws ec2 create-security-group --description "QA Sample SG" --group-name QASGSAMPLE2020
```

After running this command, the security group ID will be shown in the console, this is
important, so save the security group ID.

2. We only need to create inbound rules, as the default outbound rule is always configured.
   You will need to get the ip address of your home address, as we will be creating a rule
   to allow you to SSH from your home network.

```
aws ec2 authorize-security-group-ingress --group-id (own security group id) --protocol tcp --port 22 --cidr (own ip address)
```

Change the **own ip address** and **own security group id** to the cidr block for your ip address and security group id.

Next we will be looking at creating a launch configuration to base our AutoScaling Group.

Creating a Launch Configuration

1. We will be creating a simple launch configuration. This means when the autoscaling group decides to scale out, it can look at the launch configuration to
   create more instances to be part of the autoscaling group.

For this tutorial, we will be creating:

- Name: *"qa-sample-launch-configuration"*
- OS: *Ubuntu 18.04LTS*
- Hardware: *t2.micro*

```
aws autoscaling create-launch-configuration --launch-configuration-name qa-sample-launch-configuration --image-id ami-04137ed1a354f54c4 --instance-type t2.micro --key-name (your key name) --security-groups (your security group id)
```

You will need to use a key that you have created before and have access to the .pem key so
you may use it to connect to your instance once it launches in an autoscaling group.

Finally, we will be looking at creating our autoscaling Group.

Creating AutoScaling Group

When creating the autoscaling group through AWS CLI you will need to define the subnet ids that contain our EC2 instances.

We will need to get the VPC id so we can find out the subnet ids we want. To get the VPC id, run the following:

```
aws ec2 describe-vpcs
```

You will need to copy the vpc id as this will be needed for the next command. You **MUST**
make sure the VPC id is the same one associated with the security group for your launch
configuration. When we created the security group, we did not define a VPC, it will
by default be set to the default VPC.

Once we have the vpc id, run the following command:

```
aws ec2 describe-subnets --filters Name=vpc-id,Values=(your vpc id)
```

Copy the *SubnetId* for each subnet, as a minimum, you only need to copy 1 of the subnet ids.

Run the following command:

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name qa-sample-asg --launch-configuration-name qa-sample-launch-configuration --min-size 1 --max-size 2 --vpc-zone-identifier "(comma separated list of subnet ids)"
```

We have just created an AutoScaling Group, however there are no scaling policies associated
with the AutoScaling Group, so it will not know when to scale. We are going to create a
simple JSON file that will scale according to the CPU Utilisation.

We need to create a json file with the configuration, run the following:

```
cat << EOF > config.json
{
  "TargetValue": 80.0,
  "PredefinedMetricSpecification":
    {
      "PredefinedMetricType": "ASGAverageCPUUtilization"
    }
}
EOF
```

Now we just need attach this policy to our AutoScaling Group, run the following commands:

```
aws autoscaling put-scaling-policy --policy-name QASampleCPUUtil --auto-scaling-group-name qa-sample-asg --policy-type TargetTrackingScaling --target-tracking-configuration file://config.json
```

Test our scaling policy

1. SSH into the instance that is part of our AutoScaling Group.
2. Run the following commands to stress test our instance, therefore testing the scaling:

```
sudo apt update -y && sudo apt install stress -y
sudo stress --cpu 8 --timeout 1200
```

If you wait long enough there should be another instance spinning up. After the stress
test finishes it will terminate one of the instances as the CPU usage drops below 80%.

Clean up our environment

1. Remove all associated instances from the AutoScaling Group. We can do this by setting
   the min, max and desired capacity to 0, therefore removing all instances.

```
aws autoscaling update-auto-scaling-group --auto-scaling-group-name qa-sample-asg --min-size 0 --max-size 0 --desired-capacity 0
```

2. Delete the AutoScaling Group:

```
aws autoscaling delete-auto-scaling-group --auto-scaling-group-name qa-sample-asg
```

3. Delete the launch configuration:

```
aws autoscaling delete-launch-configuration --launch-configuration-name qa-sample-launch-configuration
```

4. Delete the Security Group:

```
aws ec2 delete-security-group --group-name QASGSAMPLE2020
```

Exercises

There are no exercises for this module.