AWS Auto Scaling Group (ASG)

One of the benefits of using a public cloud is the elasticity of its resources. Amazon Auto Scaling Group (ASG) allows cloud EC2 instances to scale (up or down) automatically by registering thresholds or conditions to ensure there is always sufficient capacity available to handle the workload. During higher demands EC2 instances are added to maintain performance requirements and during off hours instances are deducted to minimize costs. Auto scaling is ideal for workloads that experience dramatic usage variations during certain times of day, month or year. In addition to resource scaling, Auto scaling also handles server failures.

" Auto Scaling is a way to set the cloud temperature. Auto scaling Policies are used to set the thermostat to the right temperature"

An Auto Scaling policy consists of:

- A **launch configuration** that defines the servers that are created or destroyed in response to increased or decreased demand. Launch configuration consists of: AMI, instance type, and security group.
- An Auto Scaling Group (ASG) uses the launch configuration to launch or terminate EC2 instances. ASG requires a fleet of the same type of instances, ideally, doing the same task.

ASG make decisions:

- What AZ (Availability Zone) to launch a new instance
- Which ELB (Load balancer) the new instance will receive traffic
- How many instances (min/max) to run at any given time

Scaling policy can be configured to trigger:

- At some scheduled time. This is mainly done to perform routine tasks or in anticipation of load that may require longer warm up
- Static Policy: When an instance is terminated. New instance is launched to meet the minimum instance count. For example, if the launch configuration specifies one instance should always be online, then the new instance is launched to replace the failed instance.
- Dynamic Policy: At some change of state or condition by checking system and application specific metrics in Amazon Cloudwatch. Alarms are configured to inspect Amazon CloudWatch metrics, storing application and system resource usages. For example: CPU utilization, Load average (number of requests waiting for CPU), RPS (Request/second) rate. User specifies the conditions like:
    - CPU utilization or idle time stays at some threshold for the extended period. CPU idle time < 5% for 5 minutes across the auto scaling group.
    - True or False conditions

It is important not to scale up or down quickly considering the cost associated with Amazon instance usage. Amazon charges instance hourly usage and thus cost is per hour even if instance stays up for a few seconds. Thus consider scenarios listed below when making auto scaling decisions:

- Auto scale policy rules should check if a particular condition exists for a few minutes. It is common to see a huge spike that lasts for a few seconds and go away.
- CloudWatch collects events (metrics) and takes a minute to aggregate them for consumption. Auto scaling polls these CloudWatch metrics for another 60 seconds.
- Once the required condition is met and an alarm is triggered, the instance is booted. Depending on server and application configuration, it may take several minutes before the application on new instance becomes ready to serve a production traffic
- AWS recommends configuring Auto Scaling Policies to scale up quickly and scale down slowly. Thus allows applications to better respond to increased traffic loads after a scale-up event to make more economical use of the AWS hourly billing cycle.
- Instances are terminated during scale down in launch order. That means, the oldest launched instances were terminated first. Idea is to roll in new changes to application by updating your launch configuration to a newer AMI and then triggering Auto Scaling event (increase the minimum size)

If interested in receiving notification for Auto Scaling activities, configure Auto Scaling to notify you when new instances are launched or terminated using *Amazon SNS (Simple Notification Service***)**. SNS makes it easy to set up, operate, and send notifications from the cloud. It provides developers with a highly scalable, flexible and cost-effective way to publish messages from an application and immediately deliver them to subscribers or other applications.

LAB:

Register *user-data-file **"asg-cpu-metrics.sh"*** with launch configuration that gets executed after the instance is launched. We will be using the *asg-cpu-metric.sh* file below to install php and mysql server packages that will be used in setting up a web page. Create a file *"asg-cpu-metric.sh"*:

```
#!/bin/sh
# root user only
if [ "$(id -u)" != "0" ]; then
echo "This script must be run as root" 1>&2
exit 1
fi
export DEBIAN_FRONTEND=noninteractive
apt-get update
apt-get install -y mysql-server php5 php5-mysql
service apach2 restart
cd /tmp
wget http://us-east-1-aws-training.s3.amazonaws.com/self-paced-lab-4/examplefiles-as.zip
unzip examplefiles-as.zip
```

mv examplefiles-as/ /var/www/html*
rm /var/www/html/index.html

<SAVE>

Launch Configuration:
Create a new launch configuration:
$as-create-launch-config --image-id <AMI-ID> --instance-type <INSTANCE-TYPE> --group <SECURITY-GROUP> --user-data-file <FILE-NAME> --key <PEM-KEY-NAME> --launch-config <LAUNCH_CONFIG_NAME>
Example:
$ as-create-launch-config --image-id ami-00b5a124dc83b1b4b --instance-type t2.micro --group cloudperf ***--user-data-file asg-cpu-metric.sh*** *--key UCSC --launch-config* ***cloudperf-Test-Launch***

If successful in creating a launch configuration, you should see the output
OK-Created launch config

List your launch configuration.
$ as-describe-launch-configs cloudperf-Test-Launch --show-long

If you decide to delete it due to some reason, type:
$as-delete-launch-config AutoScale_Launch

Static or Fixed Auto Scaling Group (ASG):
Create a static Auto Scaling Group (ASG) that will use this launch configuration. It is a static size ASG because the number of instances at any time stays the same.

$ as-create-auto-scaling-group <ASG-NAME> --launch-configuration <LAUNCH_CONFIG_NAME> --load-balancer <LOAD-BALANCER> --min-size <MINIMUM-INSTANCES> --max-size <MAXIMUM-INSTANCES> --availability-zones <LIST-OF-ZONES-TO-USE-FOR-INSTANCES>

Example: Create an ASG “static-ASG” that uses launch configuration “cloudperf-Test-Launch” to launch 2 instances behind load-balancer “elbcli-cloudperf” spanning across multiple AZ “us-west-1”, “us-west-2”. In case of instance failure, new instance is automatically launched to keep the desired capacity to 2 instances.

$ as-create-auto-scaling-group static-ASG --launch-configuration cloudperf-Test-Launch --load-balancers elbcli-cloudperf --min-size 2 --max-size 2 --availability-zones us-west-1b, us-west-1a

If successful in creating AutoScaling group, you should see:
OK-Created AutoScalingGroup

If you decide to delete ASG due to some reason or mistake, type:
$as-delete-auto-scaling-group static-ASG --force-delete

Print detail about the ASG:
$as-describe-auto-scaling-groups static-ASG --headers
AUTO-SCALING-GROUP GROUP-NAME LAUNCH-CONFIG AVAILABILITY-ZONES LOAD-BALANCERS MIN-SIZE MAX-SIZE DESIRED-CAPACITY
AUTO-SCALING-GROUP static-ASG cloudperf-Test-Launch us-west-1b,us-west-1a elbcli-cloudperf 1 1 1
INSTANCE INSTANCE-ID AVAILABILITY-ZONE STATE STATUS LAUNCH-CONFIG
INSTANCE i-4e62408e us-west-1b InService Healthy cloudperf-Test-Launch

You can view the scaling activities by typing:
$as-describe-scaling-activities --show-long

Scaling activities should show two instances were launched to meet the requirement of ASG. Considering it is a static sized ASG (min and max are the same), there should be 2 instances running all the time. If one instance is terminated or failed, a new instance will be launched to meet ASG requirements automatically.
Terminate one instance and monitor scaling activity. New instances will be launched automatically to meet the min ASG size.

Dynamic Auto Scaling Group (ASG):
Create a dynamic auto scaling group that uses cloudwatch metrics (e.g: load average, cpu utilization, application Request/sec etc..) to scale up or down number of instances with minimum and maximum range. When dynamically scaling resources, you need to understand:

- **cooldown (***default: 300 seconds***)** period starts when scaling activity (adding or removing of ec2 resource) is in progress due to alarm trigger. During the scaling activity any new alarms will be ignored, this is to avoid adding more instances than necessary.
- **grace-period** (*default: 300 seconds*) starts after a new instance comes into service and when autoscaling health check starts. During this time, any health check failure for that instance is ignored
  http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/Cooldown.html
- **Auto Scaling Policies:**
  - scale up policy increases number of instances as result of scale up event.
  - scale down policy decreases number of instances as result of scale down event.

  NOTE: increment/decrement count can be specified as:
  - ExactCapacity: fixed capacity increase
  - ChangeInCapacity: increment to current capacity
  - PercentChangeInCapacity: percent increase in current capacity

$ as-create-auto-scaling-group dynamic-ASG --launch-configuration cloudperf-Test-Launch --load-balancers elbcli-cloudperf --min-size 2 --max-size 6 --desired-capacity 3 --availability-zones us-west-1b, us-west-1a

Where: desired-capacity is the initial count used when instances are launched in new ASG.

Create auto-scaling scale up/down policies. You can associate different cooldown period with each policy:

*$ as-put-scaling-policy **dynamic-ASG-scale-up-policy** --auto-scaling-group dynamic-ASG --adjustment=**1** --type ChangeInCapacity --cooldown 300*

Sample output: (*This output is used when associating this policy to cloudwatch metric)*
arn:aws:autoscaling:us-west-1:442122186855:scalingPolicy:4da2cae3-9a7a-41bb-8099-8c71f4cd0bad:autoScalingGroupName/dynamic-ASG:policyName/dynamic-ASG-scale-up-policy

*$ as-put-scaling-policy **dynamic-ASG-scale-down-policy** --auto-scaling-group dynamic-ASG --adjustment=**-1** --type ChangeInCapacity --cooldown 300*

Sample Output: (*This output is used when associating this policy to cloudwatch metric)*
arn:aws:autoscaling:us-west-1:442122186855:scalingPolicy:e16df9da-0c88-48f1-8e54-99a856c10d1e:autoScalingGroupName/dynamic-ASG:policyName/dynamic-ASG-scale-down-policy

CloudWatch Metrics
Amazon CloudWatch monitors EC2 resources and applications running on them in real-time. CloudWatch collects:

- System and application level metrics (cpu usage, memory, disk, net, RPS, Latency.. etc)
- Watch changes in the resource usage and then send alarms and email notifications if usage crosses some threshold or rules that you define.
- In addition to monitoring the built-in metrics that come with AWS, you can monitor your own custom metrics. With CloudWatch, you gain system-wide visibility into resource utilization, application performance, and operational health.

For example, you can monitor CPU usage and disk reads and writes of your Amazon Elastic Compute Cloud (Amazon EC2) instances and then use this data to determine whether you should launch additional instances to handle increased load. You can also use this data to stop underutilized instances to save money.

Thus in order for an auto-scaling up/down policy to trigger, you need to associate these policies to CloudWatch metrics. In this lab, we will evaluate "*CPUUtilization*" cloudwatch metrics and invoke scale-up policy to increase instance count when %cpu utilization goes over 80% or scale-down policy to decrease instance count when %cpu utilization drops down to 30%.
To avoid acting on alarm prematurely, we make sure action is taken only if CPU utilization stays at threshold levels (> 80% or < 30%) for 1 consecutive period (--evaluation-period) of 120 seconds (--period).
CPUUtilization metrics supports statistics of type: *average, sum, minimum and maximum*. For Alarm calculation, we will use *"Average"* of CPUUtilzation.

Install cloudwatch CLI: **$sudo apt-get install moncli**
Setup an alarm and associate it with scale-up policy:
$mon-put-metric-alarm HighCPU --comparison-operator GreaterThanThreshold --evaluation-periods 1 --metric-name ***CPUUtilization*** *--namespace "AWS/EC2" --period 120 --threshold* ***80*** *--alarm-actions*
arn:aws:autoscaling:us-west-1:442122186855:scalingPolicy:4da2cae3-9a7a-41bb-8099-8c71f4cd0bad:autoScalingGroupName/dynamic-ASG:policyName/dynamic-ASG-scale-up-policy --dimensions "AutoScalingGroupName=**dynamic-ASG**" --statistic Average

NOTE: ***--alarm-actions*** *takes ARN URL associated with the scale up policy(see section on creating scale up policy, as described above)*

If successful installing Alarm, you should see:
OK-Created Alarm

Setup an alarm and associate it with scale-down policy:
$mon-put-metric-alarm LowCPU --comparison-operator GreaterThanThreshold --evaluation-periods 1 --metric-name **CPUUtilization** --namespace "AWS/EC2" --period 120 --threshold **30** --alarm-actions
arn:aws:autoscaling:us-west-1:442122186855:scalingPolicy:e16df9da-0c88-48f1-8e54-99a856c10d1e:autoScalingGroupName/dynamic-ASG:policyName/dynamic-ASG-scale-down-policy --dimensions "AutoScalingGroupName=**dynamic-ASG**" --statistic Average

NOTE: ***--alarm-actions*** *takes ARN URL associated with the scale down policy(see section on creating scale down policy, as described above)*

If successful installing the Alarm, you should see:
OK-Created Alarm

Check if Alarms are installed correctly.
$mon-describe-alarms
HighCPU OK arn:aws:autoscalin...ASG-scale-up-policy AWS/EC2 CPUUtilization 120 Average 1 GreaterThanThreshold 80.0

LowCPU OK arn:aws:autoscalin...G-scale-down-policy AWS/EC2 CPUUtilization 120 Average 1 GreaterThanThreshold 30.0

You can uninstall Alarms in case you made a mistake:
$ mon-delete-alarms HighCPU
$ mon-delete-alarms LowCPU

Generate Load:

Open a URL to ELB:
http:://myelb.mydomain.net
Click “Generate Load”

This will start CPU Load on one of the instance for few minutes. This will result in HighCPU alarm to trigger scale up policy and thus scale up activity. Once the CPU Load stops, LowCPU Alarm will trigger scale down policy and thus scale down activity. You can monitor scaling activities using:
$as-describe-scaling-activities --show-long
Alarm status
$mon-describe-alarms --headers
$mon-describe-alarms --show-long
Status of ASG:
$as-describe-auto-scaling-groups dynamic-ASG --headers

You can also manually add or delete instances into the auto scaling group within the min/max range. Let’s say, if the min=1 and max=6. You can manually increase desired capacity to 3 by running:
$aws autoscaling set-desired-capacity --auto-scaling-group-name static-ASG --desired-capacity 3

References:
Static Scaling
http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/GettingStartedTutorial.html
Dynamic Scaling Based on Cloudwatch Metrics:
http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/policy_creating.html
http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-manual-scaling.html
http://www.ecloudgate.com/Doc/AutoScaling

Scaling Based on SQS Queue Size
http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-scale-based-on-demand.html

Command Line reference:
http://docs.aws.amazon.com/cli/latest/reference/autoscaling/create-launch-configuration.html
Autoscaling Cheatsheet: http://awsdocs.s3.amazonaws.com/AutoScaling/latest/as-qrc.pdf

CloudWatch Metrics:
http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/WhatIsCloudWatch.html
CloudWatch Scripts:
http://ec2-downloads.s3.amazonaws.com/cloudwatch-samples/CloudWatchMonitoringScripts-v1.1.0.zip