I, Dhaarini Chandrasekar, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:
AWS Flap Detector: An Efficient way to detect Flapping Auto Scaling Groups on AWS Cloud

Student’s name: Dhaarini Chandrasekar

This work and its defense approved by:

Committee chair: Paul Talaga, Ph.D.
Committee member: Nan Niu, Ph.D.
Committee member: Karen Davis, Ph.D.
AWS Flap Detector:
An Efficient way to detect Flapping Auto Scaling Groups
on AWS Cloud

A Thesis submitted to the
Graduate School
of University of Cincinnati
in partial fulfillments of the
requirements for the degree of

Master of Science

in the Department of Electrical Engineering and Computing Systems
of the College of Engineering and Applied Sciences
by

Dhaarini Chandrasekar
BE, Anna University, Chennai, India.
April 2014

Committee Chair: Paul Talaga, PhD
ABSTRACT

Today, large number of companies are migrating to the cloud, leaving behind the concept of maintaining traditional data centers and servers. The main reasons for this migration include reduced capital costs, reduced expenditure on infrastructure and ease of accessibility. With the increasing demand for Cloud Computing and the changing needs of users, a need to make the services on the cloud dynamic in nature is essential. However, dynamic services require constant costly updates and highly meticulous configurations.

One such dynamic service offered by Amazon Web Services (AWS) is Auto Scaling Groups (ASGs). With this service, AWS facilitates automatic scale up and scale down on the count of servers (instance resources) based on the ASG policies and conditions set by the users. A small misconfiguration or a build failure associated with the Amazon Machine Image (AMI) could cause the dynamism to occur when not actually needed. Since users are charged for the instances by the hour, unnecessary costs occur even if the usage is for as less as a minute.

This situation of unnecessary launch and termination of instances is termed as “flaps” and can be compared to oscillations in signals. To prevent energy dissipation in case of oscillating signals, damping of signals is performed.

This is similar to the problem of flapping in ASGs. We have come up with a software called AWS Flap Detector as a solution to this problem. AWS Flap Detector efficiently detects and reports flapping Auto Scaling Groups and paves the way for correction. This in turn helps prevent unnecessary resource allocation and billing.
ACKNOWLEDGEMENTS

First and foremost, my sincere thanks to my advisor Dr. Paul Talaga for being the greatest source of inspiration and encouragement throughout my journey as a graduate student. His constant support and motivation made this thesis possible. Thanks to Dr. Karen Davis and Dr. Nan Niu for graciously accepting to be on my panel committee and providing me with their valuable feedback.

I would like to thank my Manager, John Bartels from Interactive Intelligence for supporting me throughout my thesis. This thesis would have never been possible without his guidance. He will always be an inspiration to me.

If not for my family, I would not be here in the first place. Thank you for believing in me, praying for me and being there for me always. Thanks to all my friends who have provided me with enough entertainment and support that was needed.

My utmost thanks to the invisible power over us that keeps everything going.
# Table of Contents

1. **Introduction**: ................................................................................................................. 1
2. **Problem Statement** ........................................................................................................ 6
3. **Background and Definition** .......................................................................................... 8
   3.1 Damping in Signals...................................................................................................... 8
   3.2 AWS Auto Scaling Groups ......................................................................................... 10
      3.2.1 Launch Configurations ...................................................................................... 12
      3.2.2 Scaling Plans ..................................................................................................... 13
      3.2.3 Auto Scaling Group Policies .............................................................................. 14
      3.2.4 Auto Scaling Group Health Check types .......................................................... 15
   3.3 Flapping ....................................................................................................................... 18
   3.4 Causes and Effects ...................................................................................................... 22
   3.5 AWS Flap detection and Damping ............................................................................. 24
4. **Literature Review** .......................................................................................................... 26
5. **AWS Flap Detector** ...................................................................................................... 30
   5.1 Strategy and Detection Mechanism ............................................................................ 30
   5.2 The Working of AWS Flap Detector ......................................................................... 32
   5.3 The Merge Mechanism .............................................................................................. 33
6. **Validation** ...................................................................................................................... 35
   6.1 Results based on build failures (AMI) ....................................................................... 37
   6.2 Results based on Health Check types ........................................................................ 41
   6.3 Results based on Policy Thresholds .......................................................................... 44
7. **Future Work** .................................................................................................................. 48
8. **Conclusion** ..................................................................................................................... 49
9. **References** ..................................................................................................................... 50
List of figures

Figure 1: Damping in Signals ................................................................. 10
Figure 2: Scaling conditions in AWS Auto Scaling Groups ............... 18
Figure 3: The Flapping Scenario .......................................................... 20
Figure 4: Flapping condition due to a Policy Breach ......................... 21
Figure 5: Flapping condition due to a Health Check Failure ............... 21
Figure 6: AWS Flap Detector - Mechanism ......................................... 31
Figure 7: AWS Flap Detector – The Merge Mechanism ..................... 34
Figure 8: Auto Scaling Groups Consolidated Report ......................... 37
Figure 9: Case 1 – Scaling activity all day .......................................... 38
Figure 10: Case 1 - Email alert ............................................................... 39
Figure 11: Case 1 - Scaling activity from 12:55 to 14:05 ................. 40
Figure 12: Case 2 - Scaling activity all day .......................................... 42
Figure 13: Case 2 - Email alert ............................................................... 43
Figure 14: Case 2 – Scaling activity from 16:50 to 18:10 ................. 44
Figure 15: Case 3 – Scaling activity all day .......................................... 45
Figure 16: Case 3 – Email alert ............................................................... 46
Figure 17: Case 3 – Scaling activity from 10:05 to 12:10 ................. 47
1. Introduction:

From small scale companies such as financial solution providers to established organizations like Netflix and NASA, Amazon Web Services (AWS) has been the versatile solution for their infrastructure requirements. Today, more and more companies are migrating to the cloud leaving behind the concept of traditional network set up and infrastructure management in their own data centers. AWS has been the leading cloud infrastructure provider over the past few years. AWS has been offering several services for compute, storage, database and networking. One of the most noticeable and interesting offerings of AWS is the dynamic nature of its services that allow scalability of resources according to changing needs of its customers. This has helped Amazon leverage its available resources and distribute them across multiple customers. For the customers of AWS, it has helped save money by enabling them to purchase just the right amount of resources. For example, Auto Scaling Groups is a service offered by AWS that ensures that users have the correct number of Elastic Compute Cloud (EC2) instances running at any point of time to meet changing demands [1]. AWS Auto Scaling Groups make use of EC2, which is a web service that provides resizable compute capacity on AWS cloud.
The following are a few ways automatic scaling is done for the AWS Auto Scaling Groups:

• Scale on Demand
  In case of Scale on Demand, scaling of resources depends on the policies set by users during creation of their Auto Scaling Groups. Thus scale up or a scale down occurs based on certain metrics like CPU utilization, network traffic, disk usage etc.

• Fixed size scaling
  With Fixed size scaling, scaling of resources depends on the count of servers that the user wishes to have at any point of time. This count is categorized as minimum, maximum and and desired number of instances which the user specifies.

AWS associates CloudWatch in both the cases for monitoring. It scales up and down whenever the conditions are breached.

Amazon CloudWatch is a monitoring service for AWS cloud resources and other applications that run on AWS Cloud. Amazon CloudWatch is used to collect and track metrics, monitor log files and set alarm conditions. CloudWatch can monitor AWS resources such as Amazon EC2 instances, Amazon DynamoDB tables, and Amazon RDS DB instances. CloudWatch also provides monitoring of custom metrics generated by user applications and services, and other log files that the user applications generate. Amazon CloudWatch can be deployed to gain system-wide visibility into
resource utilization, application performance, and operational health. The primary goal of CloudWatch is to use these insights to react and keep the application running smoothly.

When deployed to work with Auto Scaling Groups, CloudWatch detects a scope for scale up based on either scale on demand or fixed size scaling. It then triggers a scale up as per the policy associated with the Auto Scaling Group or the min-max conditions that were set. The instances are built as per the Amazon Machine Image (AMI) specified in the launch configuration. An Amazon Machine Image contains the required information to launch an instance on the AWS Cloud. Multiple instances can be launched from one AMI. An AMI includes:

- A template for the root volume for the instance (for example, an operating system, an application server, and applications)
- Launch permissions that control which AWS accounts can use the AMI to launch instances
- A block device mapping that specifies the volumes to attach to the instance when it's launched

It takes a considerable amount of time (sometimes up to 1800 seconds) for an instance resource to launch itself and become available depending on the nature of the AMI. The AMI is subject to change depending on the dynamic nature of user requirements. In such a scenario, it is required that CloudWatch wait for a time that is greater than the time taken for an
instance to become available before it performs a second check on the Auto Scaling Group. If not, it would detect deficiency of resources, notify and trigger continuous scale up of instances. As a result, instances would be spun up continuously as CloudWatch would still be detecting a scope for scale up. Within a span of few minutes, a number of instances would be triggered for launch. This continues until the required number of instances are finally made available after which the scale down occurs. This becomes a problem when the maximum number of instances specified is high in number and the time taken for an instance to become in-service is long. Since the instances are charged on an hourly basis, this could incur a cost for customers and a resource allocation problem for AWS. This condition can be termed as an “Auto Scaling Groups’ Flap”.

Detecting such a condition is essential and requires understanding of few cloud based concepts and the working of few other services offered by AWS. Chapter 2 defines the problem statement in brief with respect to Auto Scaling Groups. Chapter 3 gives an insight into the main concepts and services which include the working of AWS Auto Scaling Groups, the parameters involved with Auto Scaling Groups and the causes for Auto Scaling Flaps. This chapter also illustrates the problems that occur with certain configurations of Auto Scaling groups with examples. There is also an insight into oscillations and damping in signals, comparing them to Auto Scaling Flaps and the purpose of AWS Flap Detector. Having
established the groundwork, Chapter 4 provides an overview of the literature on this topic, illustrating problems as well as solutions currently favored by Cloud developers and architects. Chapter 5 focuses on the product – AWS Flap Detector, its operation and working methodology. Chapter 6 describes the parameters under which AWS Flap Detector was tested and validated. A detailed analyses of the test results follow. Chapters 7 and 8 offer some suggestions for future work and conclude the discussion on AWS Flap Detector.
2. Problem Statement

With respect to Auto Scaling Group flaps, AWS suggests that users provide CloudWatch enough wait time before it can evaluate conditions for scale ups again. This time can either be the Default Cool Down Duration or the Health Check grace period depending on the Auto Scaling Group configuration and user requirements. It is required that these be high enough to accommodate launch and availability of instances before CloudWatch can perform a subsequent check and report unhealthiness or a policy breach. However, AWS doesn’t detect unnecessary launch of instances due to a low wait time for CloudWatch. It instead triggers scale ups and then eventually terminates the excess instances instead of reporting flaps to its customers. AWS encourages its customers to set up their accounts to receive notifications on change in behavior of their Auto Scaling Groups. However, there is no direct and, dedicated system for the detection of flaps.

This thesis project - AWS Flap Detection offers:

• An efficient algorithm in place for hourly detection and reporting of Auto Scaling Groups’ flaps.

• Continuous monitoring and immediate reporting via AWS SES (Simple Email Service)
Developers and customers using Auto Scaling Groups can deploy AWS Flap Detector to prevent unnecessary resource allocation due to the launch of expensive instances.
3. Background and Definition

This chapter gives an insight into the several concepts involved in the working and configuration of AWS Auto Scaling Groups. It also provides real time examples of flapping scenarios and offers a comparison to AWS Auto Scaling Groups' flaps.

3.1 Damping in Signals

Damping in signals is the process of decreasing the amplitude of an oscillation to prevent energy from being drained from the system to overcome frictional or other resistive forces [2].

The types of damping include

- Critical damping
- Over damping
- Under damping
- No damping

The Damping in signals performed to prevent energy dissipation can be explained with a real life example. Let us consider a thermostat that senses the temperature of a system and works towards maintaining the temperature around a desired temperature set by users. It does this by triggering the heating system or the cooling system on and off. This helps bring the temperature back to desired equilibrium temperature [4].

Over damping is the condition where the system returns to its equilibrium without oscillating. Considering a temperature drop in the atmosphere, the
thermostat turns off the cooling system and the equilibrium temperature is reached gradually without any fluctuations. This is an example of over damped systems.

Under damping is when the system oscillates before achieving equilibrium. When there is a temperature drop, the thermostat kicks off the cooling system in order to achieve the equilibrium temperature gradually. During this time, there is a temperature rise in the system due to external reasons, the thermostat has to turn on the cooling system. This results in oscillations until the equilibrium temperature is finally reached. This is an example of an under damped system.

A non-damped system is one that oscillates at its natural resonant frequency. If there is a continuous fluctuation in the temperature of the system due to external reasons, the thermostat works continuously on regulating the temperature endlessly. This could be a near example of a non-damped system.

Critically damped system returns to its equilibrium as soon as quickly as possible without oscillating. When the thermostat is designed to sense temperature changes quickly, it would trigger off the cooling/heating system and help reach equilibrium as soon as possible. This is an example of a critically damped system.
AWS Flap detector works towards detecting flaps in Auto Scaling Groups as quickly as possible and thereby help achieve the critically damped condition.

Figure 1 depicts the types of damping in oscillations with the working of a thermostat as an example.

![Figure 1: Damping in Signals](image)

This process of damping in oscillations is done to prevent energy dissipation in signals. The oscillations in signals can be compared to Flaps in Auto Scaling Groups and the process of Damping can be compared to the goal of AWS Flap Detector – detecting flaps and preventing unnecessary scale ups and scale downs for the Auto Scaling Groups.

### 3.2 AWS Auto Scaling Groups

Amazon Web Services offers Auto Scaling Groups as a service which enables users to have the suitable number of EC2 instances to handle the load to their application. An Auto Scaling Group is typically a collection of instances with a minimum and maximum number set to ensure that the
number of instances running is always well within that range. For example, one can set an Auto Scaling Group to have a minimum of 1 instance, a maximum of 4 and a desired number of 2 instances. The scaling occurs based on the healthiness of the instances by adjusting the number of running instances within this specified range. Auto Scaling Groups also optionally have policies associated with them telling them when to scale up and scale down depending on the user needs [1]. For example, a user might set a policy that enables launch of an instance every time the CPU utilization exceeds 75% and terminates the instance when it goes below 25%. The main purpose of this service offered by AWS is to ensure that only the required number of instances are running at all times.

Some of the notable benefits of Auto Scaling groups are:

- Better fault tolerance: Auto Scaling detects when an instance is “unhealthy”, terminates it, and launches an instance to replace it. An instance is said to be “unhealthy” if it is not in running state.
- Better availability: One can configure Auto Scaling to use multiple Availability Zones. Availability Zones are isolated location of AWS resources within one geographic location (Region). If one Availability Zone becomes unavailable, Auto Scaling enables launch of instances in another one.
- Better cost management: Auto Scaling can dynamically increase and decrease capacity as needed. Because users pay for the EC2
instances they use, they save money by launching instances when they are actually needed and terminating them when they aren't needed. This dynamic nature of Auto Scaling Groups based on health checks and policy rules ensure that users pay for only what is actually required by them.

3.2.1 Launch Configurations

A launch configuration is a template that an Auto Scaling group uses to launch EC2 instances when a scale up has to occur [1]. Every Auto Scaling Group has a Launch configuration associated with it. One Launch configuration can be associated with multiple Auto Scaling Groups. However, one Auto Scaling Group can have just one launch configuration associated with itself. A launch configuration once associated with an Auto Scaling Group cannot be edited. If required, users need to create a new Launch configuration and associate it with the Auto Scaling Group. This would be used for the next set of scale ups and scale downs. During the creation of a launch configuration, the user specifies information for the instances such as the ID of the Amazon Machine Image (AMI), the instance type, a key pair, one or more security groups, and a block device mapping. Every time a scale up occurs and new instances are to be spun up, the launch configuration typically holds information about type and nature of the instances.
3.2.2 Scaling Plans

Auto Scaling allows users to scale up and down in several ways [1].

- Maintaining a fixed instance count at all times: This allows users to configure Auto Scaling groups to maintain a minimum or specified number of running instances at all times. To maintain the current instance levels, Auto Scaling performs a periodic health check using CloudWatch on the running instances within an Auto Scaling group. When Auto Scaling finds an unhealthy instance, it terminates that instance and launches a new one. The number of instances maintained is within the range set by the user.

- Manual Scaling: This is the basic scaling methodology where the maximum, minimum, or desired capacity of the AutoScaling group is specified/updated. Auto Scaling then launches or terminates instances to match to the updated counts.

- Scaling based on a fixed schedule: In cases where the users know when metrics like CPU utilization or network traffic would go up and down, Auto Scaling allows them to specify when exactly a scale up would be required simply because that need arises on a predictable schedule. This way, the scaling actions are performed automatically as a function of time and date.

- Scale based on demand: Scale based on demand lets users define parameters that control the Auto Scaling process. For example, one
can create a policy that calls for enlarging the fleet of EC2 instances whenever the average CPU utilization rate stays above ninety percent for fifteen minutes. This serves to be very useful when one knows how to scale in response to changing conditions, but does not know when those conditions will change. Auto Scaling requires that users have two policies, one for scaling in (terminating instances) and one for scaling out (launching instances), for each event to monitor. Thus as demand increases or decreases, the correct number of instances are maintained ensuring saving of money and resources.

We would be focusing on ‘Scale on Demand’ and ‘Maintaining a fixed instance count’ for analysis as these are the two Scaling Plans that largely pave way for flapping.

### 3.2.3 Auto Scaling Group Policies

On-demand scaling or Dynamic Scaling offered by AWS requires that users define the scaling response to changing demands. This is done by the definition of policies. For example, users can specify how many instances they would like to add if the load increases by say 70% or more. They can also set conditions when the additional instances can be terminated say, when the load falls to 30% or less. Auto Scaling Group would automatically scale based on these conditions.

Auto Scaling Group uses a combination of alarms and policies to determine when the conditions for scaling are met. An alarm (Amazon
CloudWatch) is like an object that watches over a single metric (for example, the average CPU utilization of the EC2 instances in your Auto Scaling group) over a specified time period (could be set to anything from an hour to weeks). When the value of the specified metric breaches the user defined threshold during the check, the alarm sends a message to Auto Scaling, Auto Scaling then executes the associated policy to scale the group down (by terminating instances) or up (by launching instances). A policy is a set of instructions that tells Auto Scaling how exactly to respond to the alarm messages that triggered it [1]. This process continues until users delete either the scaling policies or the Auto Scaling Group itself.

3.2.4 Auto Scaling Group Health Check types

When the scaling is to depend on the min, max and desired count of instances specified by the user, AWS Auto Scaling deploys CloudWatch alarm to monitor the instances currently associated with the Auto Scaling group. For this, the user is given a choice of setting the Health Check type as either EC2 or ELB.

- With the EC2 Health Check, the CloudWatch alarm monitors the healthiness of the individual EC2 instances associated with the ASG. If the state of any of the associated instances becomes anything but ‘running’, the alarm terms them as ‘unhealthy’ and initiates launch of instances to abide by the count specified under min, max or desired.
Using EC2 Health Check for scaling is the default for AWS Auto Scaling Groups [1].

- Users can opt for ELB health checks if they have a load balancer associated with their ASGs. An Elastic Load Balancer (ELB) is used to automatically distribute incoming application traffic across multiple Amazon EC2 instances in the cloud. It enables users to achieve greater levels of fault tolerance in their applications, seamlessly providing the required amount of load balancing capacity needed to distribute application traffic. The load balancer periodically sends pings, attempts connections, or sends requests to test all the EC2 instances registered under it. Based on the response of the ping requests, it terms the instances associated with it as ‘healthy’ or ‘unhealthy’ [1].

For example, let us consider that a customer has web servers bundled in an Auto Scaling Group associated with an Elastic Load Balancer. When the Health Check is set to ELB, the load balancer would attempt to connect to the running web servers at regular intervals (Health Check Grace Period). If the load balancer is unable to reach a server under it, it terms it “unhealthy” and triggers a scale up.

After an instance has been marked unhealthy as a result of either the Amazon EC2 or Elastic Load Balancing health check, it is immediately
scheduled for replacement. And if the unhealthiness was because of anything but “terminated” state of the instance, Auto Scaling creates a new scaling activity for terminating that unhealthy instance. This is followed by a scale up activity that spins up instances such that the required/desired number of instances is met.

However, in case of ELB Health Checks, where the Load Balancer decides the healthiness of an instances associated with it, a running instance can still be termed unhealthy by the load balancer. For example, an end point hit miss by the Load balancer can cause the instances associated it to be "unhealthy" while it still might be in a running state (healthy state on EC2 level health check). This would trigger a scale up.

AWS suggests that it is totally up to the owners of the ASGs to determine how they want their Auto Scaling Groups to be scaled up in terms of Health Checks. If it is just a service that validates something readily available or fed to it, EC2 health check for scale up could be used. However, if the ASG is a web server that is load balanced, users might want to set health checks to ELBs and configure the ELB to monitor that the instances are able to be reached at all points of time. If the load balancer fails to reach the web server, the Auto Scaling group would mark it unhealthy and trigger scale ups.
Figure 2 depicts in brief, how Auto Scaling Groups work.

**Figure 2: Scaling conditions in AWS Auto Scaling Groups**

### 3.3 Flapping

There occurs a situation where a scale up event leading to launch of an instance remains under ‘pending’ state for a prolonged period of time due to the nature of AMI specified in the launch configuration of the ASG. CloudWatch alarm would typically continue to monitor and report unhealthy state of instances (if scaling on fixed count) or a threshold breach (in case of scaling on demand) of the ASG without the knowledge that the instances are in ‘pending’ state. It would further launch instances based on the associated policy when the required number of instances are already in the process of becoming available. This could become a problem if it takes too long for an instance to become available and the minimum number of instances required at all times is pretty high in number. Also when an instance that is being launched continuously fails
health checks, it paves way for repeated launch of instances and can be expensive. A number of instances could be spun up and it would take a long time (time for the required number of instances to actually become ‘in service’) for Auto Scaling to terminate all the instances that were scheduled to be launched.

In the case of a bad AMI, it would involve continuous launch and termination of instances. This is because CloudWatch would detect unhealthiness of instances trying to be launched from a bad AMI and would keep triggering its termination followed by launch of new instances. These new instances would again be launched from the same bad AMI leading to a loop like situation. In this situation, a triangle wave formation can be observed for the instance count for an Auto Scaling Group against a particular time period.

These conditions typically occur when CloudWatch alarm monitors instance states / policy breaches without waiting for a period of time that would be required for instances to be launched, healthy and “in-service”.

Figure 3 depicts the flapping scenario in AWS.

Figure 3: The Flapping Scenario
The Flap scenario due to Policy breach is depicted as under in Figure 4:

![Figure 4: Flapping condition due to a Policy Breach](image)

And, the Flapping scenario due to Health Check failures is depicted as under in Figure 5:

![Figure 5: Flapping condition due to a Health Check Failure](image)
AWS suggests that users set Default cool down duration (if scaling is on demand) and Health Check grace period (if scaling based on fixed count) for their Auto Scaling groups high enough to prevent unnecessary scheduling launch of instances. However, AWS doesn’t have a direct, dedicated method in place to detect this problem of flapping in Auto Scaling Groups.

3.4 Causes and Effects

The major reasons for flaps to occur are:

1. Time to launch an instance from an AMI being greater than Health Check Grace period or Default Cool Down Duration.

When a Health Check failure or a policy threshold breach occurs, a scale up is triggered and the launch of the required number of instance(s) takes considerable amount of time which is higher than the default cool down duration or Health Check grace period. Unaware of this, CloudWatch continuously monitors and reports potential scale ups and triggers launch of instances. This continues until the required number of instances are made available after which the additional ones are terminated. AWS charges its customers on an hourly basis for every instance they use. As a result, users pay for an entire hour for an instance that was running for less than a minute. This involves unnecessary allocation of resources and payment for an instance that was never required.
2. AMI failure.

Users change their Auto Scaling Groups to include a new Launch Configuration as a result of an updated AMI. The newly included AMI might not be stable and could cause Health Check failures. CloudWatch steps in and triggers scale ups in such a situation. However, the instances that are triggered for launch would still be launched from the same unstable AMI. This would lead to a triangle wave like situation where an instance is launched, never made available and eventually terminated. This would continue until the user manually replaces the Launch Configuration with a new one that includes a stable AMI that would not cause Health Check failures.

3. ELB Health Checks

When the Health Check type is set to ELB, launch and termination of instances are triggered whenever the load balancer reports unhealthiness. Most of the times ELB Health Check failure is because of an endpoint hit miss as opposed to instance unhealthiness. CloudWatch unmindful of this would still trigger launch of new instances after terminating existing ones that were actually healthy. This proves to be extremely costly.

When a flapping scenario occurs, it is quite likely that up to 20 instances are all scheduled for launch, resources are being allocated and AWS is
going through the process of setting up those many instances. This resource allocation is wasted once the actually required number of instances become available. The additional instances that were scheduled for launch have to be terminated. This allocation could have been put to use elsewhere if a flap had been detected at an early stage.

There are a couple of ways this can be done:

- Obtain the time taken for an instance to launch itself with AMI specified in the Launch configuration of the ASG. Set Default cool down duration and the Health Check grace period based on this calculated time.
- AWS to consider ‘in-service’ and stop cloud watch from triggering instances until ‘desired’ count is met.

### 3.5 AWS Flap detection and Damping

The issue of flapping and its detection can be compared to oscillations in signals. To control and prevent energy dissipation during such oscillations a process called damping is performed on signals.

Damping can be defined as a decrease in the amplitude of an oscillation as a result of energy being drained from the system to overcome frictional or other resistive forces [2].

Flapping in AWS Auto Scaling groups that occurs as a result of a bad AMI can be compared to non damped signals. Here, AWS would keep launching and terminating instances as a result of EC2 Health Check failures as long as the AMI remains unstable.
When the AMI is replaced with a stable one that would pass health checks, the continuous activity of launch and termination of instances would stop. This can be compared to under damped signals.

AWS Auto Scaling Groups keeps launching instances for an Auto Scaling group until the desired number of them become available. This is followed by termination of all the additional instances until the desired count is reached. In this case, there is no scope for oscillations. Equilibrium is reached after a point of time and can be compared to over damped signals.

The goal is to achieve the case of critical damping which is when the signal (Auto Scaling Group) is stabilized before any oscillations (launch and termination of instances) can occur. AWS Flap Detector works towards achieving this best case of critical damping.
4. Literature Review

Flapping has been an issue not only in the context of cloud computing but in other areas as well. For example, a thermostat in a closed environment detects a temperature drop and gradually increases the temperature of the environment. However, it takes time for the room temperature to get to the ideal temperature. If during time, there was again a temperature rise due to natural climatic reasons, the energy in increasing the temperature is wasted or overshoots [4]. This is a possible flapping scenario and has 2 dependents. The major one being time and the other being the climatic temperature. Such scenarios are hard to predict but if detected in advance, can be addressed.

The same is the case with AWS Auto Scaling. Thermostat in this case is CloudWatch and the climatic temperature can be compared to the nature of the AMI specified for use in the Launch Configuration of the ASG. If there is a time lag between the launch of an instance from an AMI before CloudWatch monitors, launch of instances is followed by their termination once the required number of instances finally become available. This ‘time’ factor and nature of the AMI play an important role.

Amazon Web Services suggests users set CloudWatch to monitor after a time interval that is more than what an instance would take to launch itself from a specified AMI [1]. But this curbs the scope for dynamism as users might want to keep updating their AMIs (launch configurations). Also AWS
notifies users on a change in behavior (launch and termination of instances) of Auto Scaling Groups but doesn’t differentiate any normal scale ups from Flaps.

Another occurrence of flapping is in routers that forwards data packets along networks [5]. Wikipedia defines this is as the situation when a router alternately advertises a destination network via one route and then advertises another (or as unavailable, and then available again) in quick sequence. A closely related term is interface flapping where an interface on a router has a hardware failure that will cause the router to announce it alternately as "up" and "down". Route flapping is caused by pathological conditions (hardware errors, software errors, configuration errors, intermittent errors in communications links, unreliable connections, etc.) within the network which cause certain reachability information to be repeatedly advertised and withdrawn. In a network where a Link-state routing protocol is run, route flapping will force frequent recalculation of the topology by all participating routers. In networks with Distance-vector routing protocols flapping routes can trigger routing updates with every state change. In both cases, they prevent the network from converging. The remedy or suggestion is that route flapping can be contained to a smaller area of the network if route aggregation is used. Since an aggregate route will not be withdrawn as long as at least one of the aggregated subnets is still valid, a flapping route that is part of an
aggregate will not disturb the routers that receive this aggregate. This idea is used in AWS Flap Detector where only aggregated information on the number of instances launched per hour is used in deciding whether an ASG is flapping. This is explained in detail in Chapter 5 under the working of AWS Flap Detector.

This problem of flapping has also been noticed by Nagios [6]. Nagios terms flapping as a condition when a service or host changes state too frequently, resulting in a sequence of problem and recovery notifications. It is stated that flapping can be indicative of configuration problems (i.e. thresholds set too low), troublesome services, or real network problems [6]. Given that Nagios can monitor any service, coming up with a reliable flap detector has been difficult.

Whenever Nagios checks the status of a host or service, it will check to see if it has started or stopped flapping. It does this by:

- Storing the results of the last 21 checks of the host or service
- Analyzing the historical check results and determine where state changes/transitions occur
- Using the state transitions to determine a percent state change value (a measure of change) for the host or service
- Comparing the percent state change value against low and high flapping thresholds
A host or service is determined to have started flapping when its percent state change first exceeds a high flapping threshold. A host or service is determined to have stopped flapping when its percent state goes below a low flapping threshold (assuming that is was previously flapping). This solution is highly probabilistic as it entirely depends on previous states rather than a fixed detection mechanism. This doesn’t support dynamism which is the backbone of today’s computing.

The studies covered here clearly indicate that flapping doesn’t have a dedicated and efficient detection mechanism in place thought its prevalence has been identified largely in many different fields.
5. AWS Flap Detector

AWS Flap Detector is a software that is designed to carefully detect and alert flapping scenarios in AWS Auto Scaling Groups. It is developed in Python and makes use of stack and list data structures. It has its own experimentally developed Flap Detection Algorithm which is tested to accurately detect and report flaps better and in a more dedicated fashion when compared to CloudWatch alerts on changing Auto Scaling Groups.

5.1 Strategy and Detection Mechanism

The idea behind AWS Flap Detector is to detect flapping Auto Scaling Groups and thereby avoid unnecessary allocation of expensive resources that would never be used. AWS Flap Detector uses an efficient mechanism to differentiate flaps from normal scale ups by:

1. Considering only a scale_up_per_hour_count greater than or equal to 5 as a flap.
2. The scale_up_per_hour_count is calculated based on a time gap of 10 seconds.

Scale ups that were triggered only after 10 seconds of a previous scale up contribute towards incrementing the scale counter. This is because scale ups that occur at the same time are a result of the same policy breach or health check failure.

AWS Flap Detector makes use of AWS CLI and Boto in Python and runs locally to fetch details of Auto Scaling groups from the user’s AWS
accounts to compute the scale count and detect flaps.

Figure 6 gives a quick overview of the mechanism behind AWS Flap Detector.

![Figure 6: AWS Flap Detector - Mechanism]

The first step is to fetch all the Auto Scaling Group names associated with the account taken for inspection. Then, for each of the ASGs from the list, the Auto scaling activities is fetched. From this activity history, the ‘start-time’ and ‘end-time’ for every associated scale ups are extracted. The next step is to filter the list to include only the ASGs that went through a scale up in the last hour. This is done by checking the time difference between
the time of scale up and the current time in seconds. The filtered list is
then passed onto a method that groups scale ups that were triggered in a
time difference of 10 seconds. This is a buffer time as scale ups triggered
at such a short interval, are generally a result of the same policy or
condition set by the user for the ASG. The start time of 2 different scale
ups are compared and if the difference is less than 10 seconds, only the
latest scale ups are retained in the Python list and the older ones are
popped out. If not, the local Python list is appended with the additional
scale up data. This updated list is passed on to a function that, depending
on the size of the list, categorizes the ASGs as safe, Possibly Flapping
(Warning) and Flapping (Alert). AWS Flap Detector then fetches the owner
of the ASG from the tag associated with the ASG/instances/AMI used and
as the final step, send out e-mails to the owners via AWS SES with
information of their flapping ASGs.

5.2 The Working of AWS Flap Detector

The general working of AWS Flap Detector is as follows:

1. Obtain the Auto Scaling Group activity list for every Auto Scaling
   Group.
2. Filter the list to retain only the scale ups.
3. Further filter the list to retain only the scale ups that occurred in the
   last hour.
4. Combine scale ups that occurred in a span of 10 seconds to count
just as one as they might be as a result of the same policy.

5. Calculate the average scale up time as the average difference between the times of 2 distinct scale ups.

6. Categorize the Auto Scaling Group as Safe / Warning / Alert depending on the scale count of
   
   • scale_up_per_hour_count <= 3
   • 3 < scale_up_per_hour_count < 5
   • scale_up_per_hour_count >= 5

   The count "4" was arrived at based on the default cool down duration. This is generally 600 for most of the ASGs and 300 for a few. The criteria for flap alert is ((3600/600)-1) and ((3600/600)-2) is the criteria for a possible flap alert. These numbers are suitable for ASGs with default cool down duration of 300 too and were verified in production.

7. Send out e-mail alerts to users for the categories ‘warning’ and ‘alert’

These steps bundled together is AWS Flap Detector’s working mechanism. AWS Flap Detector runs every hour on weekdays on real time Auto Scaling Groups.

5.3 The Merge Mechanism

The crux of the algorithm to detect flapping in AWS Flap Detector lies with the way the scale ups are actually counted to detect flaps. That is, if 2
consecutive scale ups occurred in a time difference of 10 seconds, they are counted as ‘1’ as they are generally a result of the same Auto Scaling policy. This is the Merge mechanism adopted by AWS Flap Detector to accurately detect flaps.

Figure 7 depicts the merge mechanism.

![Flowchart](image)

**Figure 7: AWS Flap Detector – The Merge Mechanism**

Based on the scale count, categorization of safe, possibly flapping and flapping alert. E-mail notifications via SES are then sent to owners for scale count of greater than 3.
6. Validation

Flapping occurs not only in AWS Auto Scaling Groups but in a wide area of subjects including Signals and Systems, electronic devices etc. Though these issues have been discussed and addressed, flapping as a scenario does not have any single and unique detection mechanism or solution. AWS Flap Detector is highly inspired by mechanisms that were brought in place for flapping in other fields like Computer networks and Signals and Systems. It provides a dedicated flap detection and reporting mechanism and is validated in various test scenarios on real time AWS Auto Scaling groups.

The principle behind validation of AWS Flap Detector is to capture cases where flapping occurred due to:

1. AMI failures
   Flaps due to AMI failures occur when instances are launched from an unstable or bad AMI that would lead to Health Check failures. This in turn triggers a continuous launch and termination of instances. Section 6.3 describes this as Case 1 captured from the real time tests conducted.

2. Low Default Cool Down Duration/Health Check Grace period
   Auto Scaling Groups flap when CloudWatch performs check on policies or healthiness before the required instances are made available. This leads to a continuous launch of instances until the
desired number of instances are finally made available. This is captured as Case 2 and Case 3 in Sections 6.4 and 6.5.

The goal of AWS Flap Detector is to detect flaps successfully, compare the results in each of the cases to determine the most frequent reason for a flap to occur. The tests were carried out for a duration of 3 months, everyday on an hourly basis on 120+ live Auto Scaling Groups each differing in their launch configurations, policy for scaling, etc. These real time Auto Scaling Groups used for our validation were used to launch instances for several services deployed by a product based company head quartered in Indianapolis. Some of these required having dedicated machines for database, publishers, log analysis etc. The tests on these Auto Scaling Groups were carried out across multiple AWS accounts to obtain a wide set of test results to accurately detect flaps and then categorize them based on the reason of occurrence. AWS Flap Detector is coded in such a way that a scale count of more than 3 would send in a failure alert. AWS Flap Detector is also programmed to send individual alert e-mails to owners of flapping Auto Scaling groups.

For an easy comparison of the several Auto Scaling Groups, we built a logging system that would fetch details of every Auto Scaling Group including the Policy, Health Check Type, Scaling Metric, Scale up and Scale down threshold and the owner of the ASG. This was run every week
as a cron job in order to obtain an updated list of the Auto Scaling groups and their details.

Figure 8 lists an example of the Auto Scaling Groups and their policies which was the output of the logging system.

### AWS Flap Detector runs on a dedicated AWS instance. We used Jenkins, an open source automation software to run AWS Flap Detector every hour from 9 am to 5 pm on weekdays.

#### 6.1 Results based on build failures (AMI)

AMI failures were one of the most frequent reasons for an Auto Scale flap to occur, as detected by AWS Flap Detector. As AMIs are always updated for better performance, patch updates etc., an AMI may be unstable and can cause an instance to fail health checks.

<table>
<thead>
<tr>
<th>ASG Name</th>
<th>Alarm Name</th>
<th>Cloud ID, Health Check Interval, Type, Owner Contact</th>
<th>Status</th>
<th>Scale Down</th>
<th>Scale Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecretBash-ASG-AUTOTN1-s30</td>
<td>SecretBash-ASG-AUTOTN1-ami-2018</td>
<td>60 EC2</td>
<td>cr, micro</td>
<td>mask <a href="mailto:janvier@rno.com">janvier@rno.com</a></td>
<td>CRU</td>
</tr>
<tr>
<td>add simulation-ASG-AUTOTN1-ami</td>
<td>add simulation-ASG-AUTOTN1-ami-2017</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>agent schedule-ASG-AUTOTN1-ami</td>
<td>agent schedule-ASG-AUTOTN1-ami-2017</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>N/A</td>
</tr>
<tr>
<td>another ASG-DB2431341434-ami</td>
<td>another ASG-DB2431341434-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-2M34458-ami</td>
<td>another agent-ASG-2M34458-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-3M45689-ami</td>
<td>another agent-ASG-3M45689-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-4M56789-ami</td>
<td>another agent-ASG-4M56789-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-5M67890-ami</td>
<td>another agent-ASG-5M67890-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-6M78901-ami</td>
<td>another agent-ASG-6M78901-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-7M89012-ami</td>
<td>another agent-ASG-7M89012-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-8M90123-ami</td>
<td>another agent-ASG-8M90123-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-9M01234-ami</td>
<td>another agent-ASG-9M01234-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-10M12345-ami</td>
<td>another agent-ASG-10M12345-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-11M23456-ami</td>
<td>another agent-ASG-11M23456-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-12M34567-ami</td>
<td>another agent-ASG-12M34567-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-13M45678-ami</td>
<td>another agent-ASG-13M45678-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-14M56789-ami</td>
<td>another agent-ASG-14M56789-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-15M67890-ami</td>
<td>another agent-ASG-15M67890-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-16M78901-ami</td>
<td>another agent-ASG-16M78901-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-17M89012-ami</td>
<td>another agent-ASG-17M89012-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-18M90123-ami</td>
<td>another agent-ASG-18M90123-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-19M01234-ami</td>
<td>another agent-ASG-19M01234-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
<tr>
<td>another agent-ASG-20M12345-ami</td>
<td>another agent-ASG-20M12345-ami-2019</td>
<td>60 EC2</td>
<td>cr, large</td>
<td>DocCloud/DevOpsManagement@rno.com</td>
<td>CRU</td>
</tr>
</tbody>
</table>

**Figure 8: Auto Scaling Groups Consolidated Report**

AWS Flap Detector runs on a dedicated AWS instance. We used Jenkins, an open source automation software to run AWS Flap Detector every hour from 9 am to 5 pm on weekdays.
This case can be compared to under damped and non damped signals. Auto Scaling groups stops the launch and termination of instances if the AMI is replaced leading to an under damped situation. If the AMI isn’t replaced, The Auto Scaling Group continues to launch and terminate instances continuously resulting in a non damped condition.

AWS Auto Scaling Flap Detector successfully detects the triangle-wake-like oscillating condition (continuous launch and termination of instances) and paves way for critical damping.

Figure 9 is a graph that shows the instance count over a time period for the Auto Scaling Group referred to as Case 1, caused due to a bad AMI.

![Graph showing Case 1 – Scaling activity all day](image-url)
At 2 pm, the relative instance count went up to 20. It can be seen from the e-mail alert at 2 pm that the count of instances for the last hour (1 pm to 2 pm) was 20.

![Email Alert](image)

Flapper detected that the following ASG of yours is probably flapping in dev.

**Alert**
- ASG Name: workforce-management-ASG-DOHXTKB4ML3v689
- Average time for scaling: 63.55 seconds
- Scale count in the last hour: 20
- AMI used: ami-81752084

For some information on Autoscaling flaps, visit Our wiki link and for details on Health Checks for ASGs visit Health Checks for ASGs on our wiki

PS: Please ignore this e-mail if the scale ups were intended by you

---

**Figure 10: Case 1 - Email alert**

Upon investigation of this Auto Scaling group and the exact pattern for scale up, it was observed that the Auto Scaling Group was continuously scaling up and down. According to the launch configuration of this Auto Scaling Group, the minimum number of instances to be running at any point of time was 2 and the maximum is 8. The desired number of instances (our threshold) was 2.

After an update of the launch configuration with a new AMI, the Auto Scaling group underwent a scale up with 2 new instance additions before the older ones were removed. Since the new instances took some time to become available before CloudWatch could check again, launch of 2 other...
instances was triggered. However these instances never became available due to the bad nature of AMI leading to a triangular wave formation. This process continued until the launch configuration was reverted back to the older one manually within the same hour. This can be compared to under damping of signals.

The graph in Figure 11 depicts the scale up and scale down that occurred from 1 pm to 2 pm for the Auto Scaling Group marked as Case 1.

![Graph](image)

**Figure 11: Case 1 - Scaling activity from 12:55 to 14:05**

If there wasn’t a revert to the old launch configuration, there would have been non damped signals situation for an infinite period. For Case 1, the occurrence of events from 1 pm to 2 pm would have continued forever.
AWS Flap detector successfully detected and reported a flap condition here and paved way for a critically damp condition as opposed to an underdamped condition. This helped in resource allocation for AWS and correction of a faulty configuration of the Auto Scaling Group for the user.

6.2 Results based on Health Check types

When the Health Check is the deciding factor for scale ups, a health check before the availability of the instances leads to continuous launch of instances without termination of any of them. Once the required number of instances are made available, AWS Auto Scaling Groups terminates the additional instances. This situation can be compared to over damping of signals.

The graph in Figure 12 depicts the instance count over a time period for the Auto Scaling Group marked as Case 2. Case 2’s Auto Scaling Group was configured to scale up based on Health Check Types.
The Auto Scaling Group was configured to have a minimum of 4 instances and a maximum of 8 instances at any point of time and the desired count of instances was also 4. AWS Flap Detector detected that at 6 pm on one of the days of test, the scale count per hour went to 6. Though this was within the permissible levels as the max count was 8, the scale ups were unnecessary as the scale ups were a result of ELB Health Check failure. The time taken for new instances to become available was less than the time the second health check was performed resulting in launch of 2 extra instances. Once the first 4 were made available, the other 2 were terminated. As a result, 6 instances were launched when only 4 were
required.

Figure 13 is an image of the e-mail alert obtained from AWS Flap Detector.

![ASG Flap Detector: Your Autoscaling Group is probably flapping!]

Flappler detected that the following ASG of yours is probably flapping in dev.

**ALERT**

ASG Name: donut-ASG-1XINTWK44A9NU-v576  
Average time for scaling: 428.5 seconds  
Scale count in the last hour: 6  
AMI used: ami-656c3ed0

For some information on Autoscaling flaps, visit [Our wiki link](#) and for details on Health Checks for ASGs visit [Health Checks for ASGs on our wiki](#).

PS: Please ignore this e-mail if the scale ups were intended by you

**Figure 13: Case 2 - Email alert**

Upon investigation of the pattern in scale count, it was observed that due to the unhealthiness of instances, new instances were triggered for launch. However, before they could become available, CloudWatch triggered launch of 2 more instances as the Health Check Grace period of this ASG was 600 which was less than the time the new instances took to become ready and in-service for the Auto Scaling group. This is depicted in the graph in Figure 14.

This can be compared to over damping of signals where the equilibrium is reached without any oscillations. AWS Flap Detector paved for critical damped situation to occur from an over damped situation.
6.3 Results based on Policy Thresholds

Most of the times, Auto Scaling Groups are configured with policies that lead to scale up based on threshold conditions for CPU utilization, Network traffic etc. Upon breach of these conditions, instance launch is triggered. Until the required number of instances becomes available, instances are launched continuously after which they are terminated. This can be compared to over damping of signals.

The graph in Figure 15 depicts flapping that occurred for the Auto Scaling Group marked as Case 3 due to policy threshold breach over a period of
Figure 15: Case 3 – Scaling activity all day

This Auto Scaling Group was configured to have a minimum of 2 instances and a maximum of 8 instances associated with it at any time and the desired count was set to 2. The policy threshold set was that of CPU utilization where a CPU utilization of more than 60 percent, would trigger a scale up and a CPU utilization of less than 25 percent would trigger a scale down. Figure 16 shows the mail alert sent by AWS Flap Detector notifying the owner of the Auto Scaling Group that there was a possible flap.
Due to the policy threshold breach, new instances (2 instances every 10 minutes) were being launched. However, before they could become available CloudWatch detected scarcity of instances and triggered launch of instances again. This led to a scale up count of 8. Once the required number of instances became available, the instance count was brought back to the suitable count of 4.

The graph in Figure 17 depicts the over damped situation where instances were launched and the additional ones terminated to maintain the suitable count at that point of time.

**Figure 16: Case 3 – Email alert**
Figure 17: Case 3 – Scaling activity from 10:05 to 12:10

This over damped situation occurred for a few minutes and was brought back to a critically damped situation by a timely flap detection by AWS Flap Detector.
7. Future Work

AWS Flap Detector successfully detects and reports flapping Auto Scaling Groups. This enables users to spend for what they actually use instead of spending for what they might need. In this section we provide suggestions and scope for future work. Currently, AWS Flap Detector makes use of a dedicated algorithm to detect and report flapping Auto Scaling Groups. A future area of improvement could be to compare the Default Cool Down Duration / Health Check Grace period with the time taken for an instance to become in service and provide a suggested correction in the wait time. However, a change in launch configuration with an updated AMI may lead to frequent recalculations adding to this solution’s overhead. Another area of improvement could be to include the activity log analysis for an accurate determination of the reason for an Auto Scaling Flap. An automatic fix feature could be added to the product that would accurately detect the cause of a flap from the obtained activity logs and make the changes to prevent a future flap.
8. Conclusion

AWS Flap Detector is a dedicated Flap detection software focusing on detection and categorization of flapping scenarios occurring in AWS Auto Scaling Groups. When AWS Flap Detector is deployed, users can be ensured that they have healthy Auto Scaling Groups with the required number of instances. Using AWS Flap Detector, users can make sure that they pay only for what they actually need. AWS Flap Detector has proved to be effective in detecting flaps accurately and has contributed towards the example organization’s money. The working solution of AWS Flap Detector can be compared to achieving a critically damped condition for oscillating signals from an over damped, under damped or non damped condition. AWS Flap Detector thereby helps to save unnecessary resource allocations and billing. Developers can put it to their advantage and focus on their applications and scaling their services. AWS Flap Detector takes over that responsibility of ensuring that users have the appropriate number of running instances at all times and are spending wisely.
9. References


[9] David Morin. Harvard University, Oscillations – Chapter 1 (http://www.people.fas.harvard.edu/)

[11] Divyakant Agrawal, Amr El Abbadi, Sudipto Das, and Aaron J. Elmore. Database Scalability, Elasticity, and Autonomy in the Cloud, University of California at Santa Barbara, Santa Barbara, California, USA.