Table of Contents

Autoscaling....2

Types of scaling....2

Autoscaling Solutions....3

1. Autoscaling with Docker....3
2. Autoscaling with AWS....3
3. Autoscaling with Azure Autoscale....4
4. Autoscaling with Horizontal Pod Scaling Kubernetes....6
5. Autoscaling with "Autoscaling Group of Instances" Google Cloud Platform....7
6. Autoscaling with Cluster Autoscaler Google Cloud Platform....8

Instances Types AWS....8

1. General Purpose....8
2. Compute Optimized....9
3. Memory Optimized....9
4. Accelerated Computing....10
5. Storage Optimized....10

Instances Types Table....11

Instances Types LRZ (Mapping from AWS)....11

Autoscaling Decision Metrics (used for Autoscaling)....12

Scaling Adjustment or Tuning Parameters....13

1. AWS....13
2. Kubernetes....14
3. Google Cloud Platform....14
4. Google Cloud Platform Cluster Autoscaler....14

Autoscaling Policies....14

1. AWS....14
2. Google Cloud Platform....16

References....17

Autoscaling

Autoscaling [1] is a method used in cloud computing, whereby the amount of computational resources in a server farm, typically measured in terms of the number of active servers, scales automatically based on the load on the farm.

In detail, Autoscaling [2] is the process of dynamically allocating the resources required by an application to match performance requirements and satisfy service-level agreements (SLAs), while minimizing runtime costs. As the volume of work grows, an application may require additional resources to enable it to perform its tasks in a timely manner. As demand slackens, resources can be de-allocated to minimize costs, while still maintaining adequate performance and meeting SLAs. Autoscaling takes advantage of the elasticity of cloud-hosted environments while easing management overhead. It does so by reducing the need for an operator to continually monitor the performance of a system and make decisions about adding or removing resources.

Types of scaling

Scaling typically takes one of the following two forms [2]:

- **Vertical** (often referred to as *scaling up and down*). This form requires that you modify the hardware (expand or reduce its capacity and performance), or redeploy the solution using alternative hardware that has the appropriate capacity and performance. In a cloud environment, the hardware platform is typically a virtualized environment. Unless the original hardware was substantially overprovisioned, with the consequent upfront capital expense, vertically scaling up in this environment involves provisioning more powerful resources, and then moving the system onto these new resources. Vertical scaling is often a disruptive process that requires making the system temporarily unavailable while it is being redeployed. It may be possible to keep the original system running while the new hardware is provisioned and brought online, but there will likely be some interruption while the processing transitions from the old environment to the new one. It is uncommon to use Autoscaling to implement a vertical scaling strategy.
- **Horizontal** (often referred to as *scaling out and in*). This form requires deploying the solution on additional or fewer resources, which are typically commodity resources rather than high-powered systems. The solution can continue running without interruption while these resources are provisioned. When the provisioning process is complete, copies of the elements that comprise the solution can be deployed on these additional resources and made available. If demand drops, the additional resources can be reclaimed after the elements using them have been shut down cleanly. Many cloud-based systems, support automation of this form of scaling.

Autoscaling Solutions

1. Autoscaling with Docker

To automate the scaling procedure with Docker, Docker Remote API [3] is used. It creates a Replicator docker image that listens to requests with container ID as the parameter and can create and deploy new docker images like the one with the given container ID. This can be used to scale the service as the traffic increases and further application can be deployed with Docker Swarm [4].

2. Autoscaling with AWS

AWS Auto Scaling [1] helps to maintain application availability and allows to scale Amazon EC2 capacity up or down automatically according to conditions defined. Auto Scaling can be used to ensure that the desired number of Amazon EC2 instances are running. It can also automatically increase the number of Amazon EC2 instances during demand spikes to maintain performance and decrease capacity during lulls to reduce costs. Auto Scaling is well suited both to applications that have stable demand patterns or that experience hourly, daily, or weekly variability in usage.

A brief description of how it works [5]:

First a collection of EC2 instances is created, called *Auto Scaling groups*. Then the minimum number of instances in each Auto Scaling group is specified, and Auto Scaling ensures that the group never goes below this size. Also, the maximum number of instances in each Auto Scaling group is specified, and Auto Scaling ensures that your group never goes above this size. The desired capacity can also be specified, and Auto Scaling ensures that your group has this many instance. Based upon the scaling policies specified, Auto Scaling can launch or terminate instances as demand on your application increases or decreases.

For example, the following Auto Scaling group has a minimum size of 1 instance, a desired capacity of 2 instances, and a maximum size of 4 instances. The scaling policies that you define adjust the number of instances, within your minimum and maximum number of instances, based on the criteria that you specify.

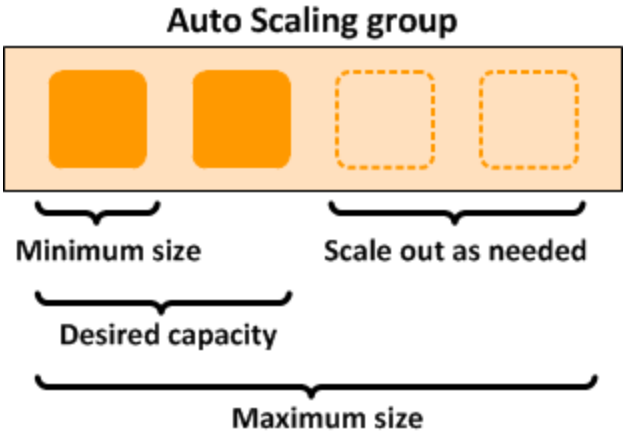

Figure 1: AWS Autoscaling Example [5]

3. Autoscaling with Azure Autoscale[1]

Azure Autoscale [6] allows to have the right amount of resources running to handle the load on the application. It allows user to add resources to handle increases in load and save money by removing resources that are sitting idle. User specify a minimum and maximum number of instances to run and add or remove VMs automatically based on a set of rules. Having a minimum makes sure the application is always running even under no load. Having a maximum limits your total possible hourly cost. Application automatically scale between these two extremes using rules you create.

Below figure describes the Autoscale:

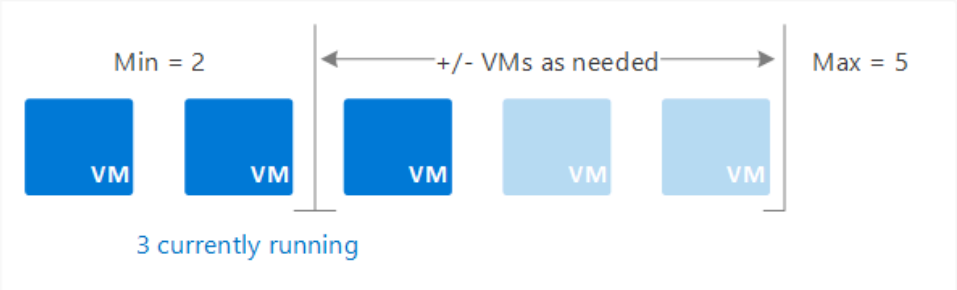

Figure 2: Azure Autoscale Example [6]

When rule conditions are met, one or more Autoscale actions are triggered. You can add and remove VMs, or perform other actions. The following conceptual diagram shows this process.

[1] Azure Monitor Autoscale applies only to Virtual Machine Scale Sets, Cloud Services, and App Service - Web Apps.

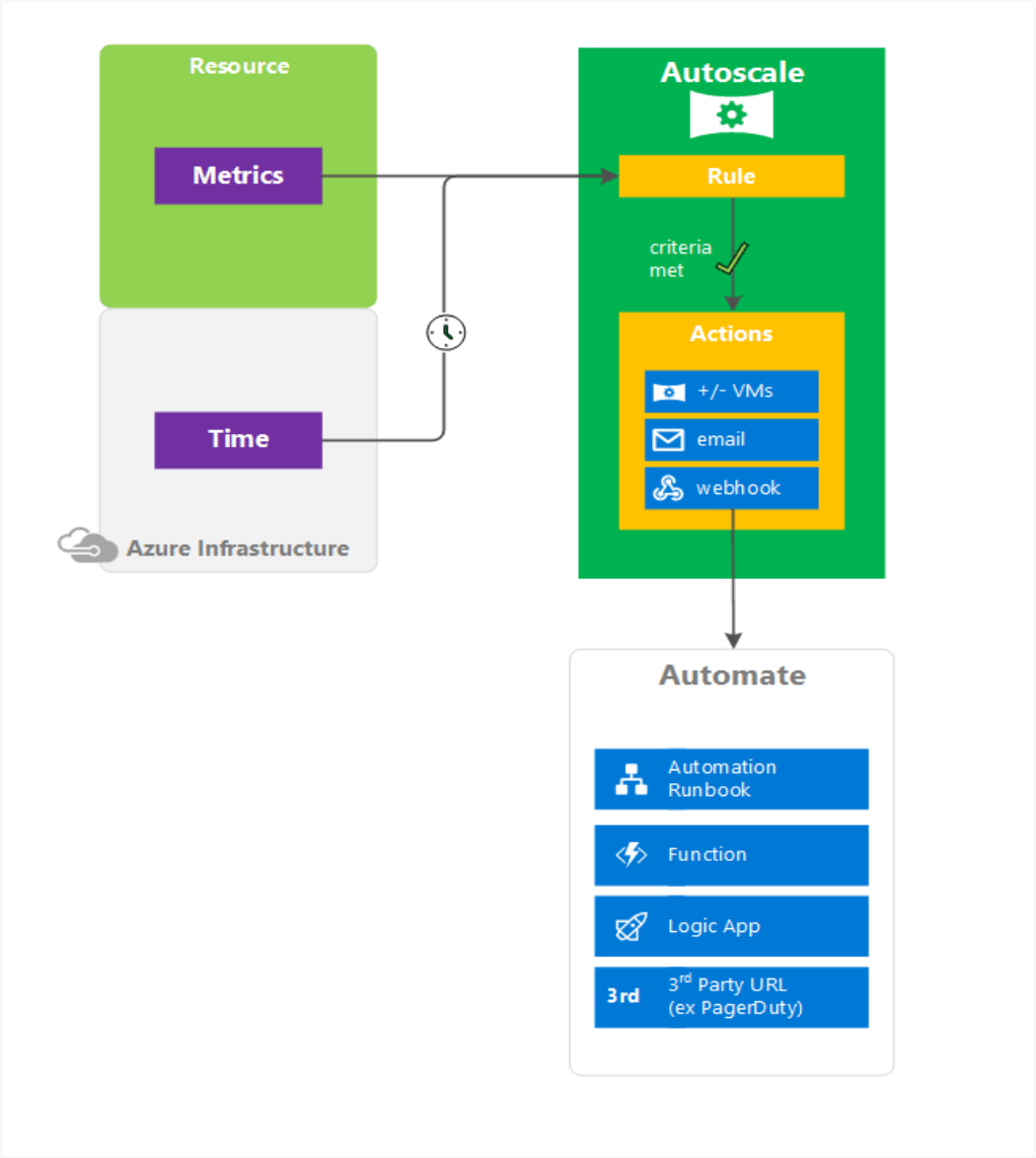

Figure 3: Azure Autoscale Process [6]

Autoscale only scales horizontally, which is an increase ("out") or decrease ("in") in the number of VM instances. Horizontal is more flexible in a cloud situation as it allows you to run potentially thousands of VMs to handle load.

In contrast, vertical scaling is different. It keeps the same number of VMs, but makes the VMs more ("up") or less ("down") powerful. Power is measured in memory, CPU speed, disk space, etc. Vertical scaling has more limitations. It's dependent on the availability of larger hardware, which quickly hits an upper limit and can vary by region. Vertical scaling also usually requires a VM to stop and restart.

4. Autoscaling with Horizontal Pod Scaling Kubernetes

With Horizontal Pod Autoscaling [7], Kubernetes automatically scales the number of pods[2] in a replication controller[3], deployment or replica set based on observed CPU utilization (or, with alpha support, on some other, application-provided metrics).
The Horizontal Pod Autoscaler is implemented as a Kubernetes API resource and a controller. The resource determines the behavior of the controller. The controller periodically adjusts the number of replicas in a replication controller or deployment to match the observed average CPU utilization to the target specified by user.

A brief detail about the working of Autoscaler:

The Horizontal Pod Autoscaler is implemented as a control loop, with a period controlled by the controller manager's **--horizontal-pod-autoscaler-sync-period** flag (with a default value of 30 seconds).

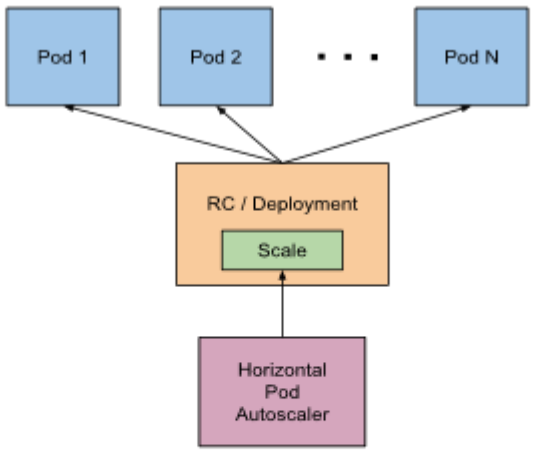

Figure 4: Kubernetes Autoscaler [7]

During each period, the controller manager queries the resource utilization against the metrics specified in each HorizontalPodAutoscaler definition. The controller manager obtains the metrics from either the resource metrics API (for per-pod resource metrics), or the custom metrics API (for all other metrics).

---

[2] A Pod is the basic building block of Kubernetes–the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents a running process on your cluster.

[3] A Replication Controller ensures that a specified number of pod "replicas" are running at any one time. More information here

- For per-pod resource metrics (like CPU), the controller fetches the metrics from the resource metrics API for each pod targeted by the HorizontalPodAutoscaler. Then, if a target utilization value is set, the controller calculates the utilization value as a percentage of the equivalent resource request on the containers in each pod. If a target raw value is set, the raw metric values are used directly. The controller then takes the mean of the utilization or the raw value (depending on the type of target specified) across all targeted pods, and produces a ratio used to scale the number of desired replicas.
  If some of the pod's containers do not have the relevant resource request set, CPU utilization for the pod will not be defined and the Autoscaler will not take any action for that metric.

- For per-pod custom metrics, the controller functions similarly to per-pod resource metrics, except that it works with raw values, not utilization values.

- For object metrics, a single metric is fetched (which describes the object in question), and compared to the target value, to produce a ratio as above.

The HorizontalPodAutoscaler controller can fetch metrics in two different ways: direct Heapster[4] access, and REST client access.

When using direct Heapster access, the HorizontalPodAutoscaler queries Heapster directly through the API server's service proxy sub resource. Heapster needs to be deployed on the cluster and running in the kube-system namespace.

The Autoscaler accesses corresponding replication controller, deployment or replica set by scale sub-resource. Scale is an interface that allows to dynamically set the number of replicas and examine each of their current states.

5. Autoscaling with "Autoscaling Group of Instances" Google Cloud Platform

Google Cloud Platform Managed instance group[5] offer Autoscaling capabilities [8] that allows to automatically add or remove instances from a managed instance group based on increases or decreases in load. Autoscaling helps applications gracefully handle increases in traffic and reduces cost when the need for resources is lower. The only thing need to define is the Autoscaling policy and the Autoscaler performs automatic scaling based on the measured load.
Autoscaling works by scaling up or down your instance group. That is, it adds more instances to your instance group when there is more load (upscaling), and removes instances when the need for instances is lowered (downscaling).

[4] Heapster enables Container Cluster Monitoring and Performance Analysis. For more information here
[5] A managed instance group uses an instance template to create a group of identical instances. You control a managed instance group as a single entity.

6. Autoscaling with Cluster Autoscaler Google Cloud Platform

Cluster Autoscaler [9] enables users to automatically resize clusters so that all scheduled pods have a place to run. If there are no resources in the cluster to schedule a recently created pod, a new node is added. On the other hand, if some node is underutilized and all pods running on it can be easily moved elsewhere then the node is deleted. This feature allows users to pay only for resources that are needed and get new resources when the demand increases.

The feature should be used when the cluster administrator wants the cluster to be automatically resized and can tolerate some disruptions that may happen when a node is deleted (for example, a replicated pod may not be running until it is scheduled again), unless part of a replica set of two or larger.

A brief description of it works:
Cluster Autoscaler periodically checks whether there are any pods waiting for a node with free resources and are not being scheduled. If such pods exist, and the Cluster Autoscaler determines that resizing a node pool would allow the pod to be scheduled, then a node pool resize is executed.
Cluster Autoscaler also monitors the usage of all nodes. If a node is not needed for an extended period and all its pods can be easily scheduled elsewhere then the node is deleted.
Cluster Autoscaler is configured per node pool. The user specifies the minimal and maximal size of each node pool and Cluster Autoscaler makes rescaling decisions within these boundaries. If the current cluster size is greater than the specified maximum, then Cluster Autoscaler waits until a new node is needed or a node can be safely deleted.

Instances Types AWS

Amazon EC2 provides a wide selection of instance types [10] optimized to fit different use cases.

1. General Purpose

- **T2**
  T2 instances are Burstable Performance Instances that provide a baseline level of CPU performance with the ability to burst above the baseline. The baseline performance and ability to burst are governed by CPU Credits. Each T2 instance receives CPU Credits continuously at a set rate depending on the instance size. T2 instances accrue CPU Credits when they are idle, and use CPU credits when they are active. *T2 instances are a good choice for workloads that don't use the full CPU often or consistently, but occasionally need to burst (e.g. web servers, developer environments and databases).*

- **M4**
  M4 instances are the latest generation of General Purpose Instances. This family provides a balance of compute, memory, and network resources, and it is a good choice for many applications.

- **M3**
  This family includes the M3 instance types and provides a balance of compute, memory, and network resources, and it is a good choice for many applications

These are good choice for Small and mid-size databases, data processing tasks that require additional memory, caching fleets, and for running backend servers for SAP, Microsoft SharePoint, cluster computing, and other enterprise applications.

2. Compute Optimized

- **C4**
  C4 instances are the latest generation of Compute-optimized instances, featuring the highest performing processors and the lowest price/compute performance in EC2.

- **C3**
  C3 instances are also the latest generation of Compute-optimized instances, featuring the highest performing processors and the lowest price/compute performance in EC2.

These are good choice for High performance front-end fleets, web-servers, batch processing, distributed analytics, high performance science and engineering applications, ad serving, MMO gaming, and video-encoding.

3. Memory Optimized

- **X1**
  X1 Instances are optimized for large-scale, enterprise-class, in-memory applications and have the lowest price per GiB of RAM among Amazon EC2 instance types.
  X1 instances are recommended for running in-memory databases like SAP HANA, big data processing engines like Apache Spark or Presto, and high performance computing (HPC) applications.

- **R4**
  R4 instances are optimized for memory-intensive applications and offer better price per GiB of RAM than R3.

R4 instances are recommended for high performance databases, data mining & analysis, in-memory databases, distributed web scale in-memory caches, applications performing real-time processing of unstructured big data, Hadoop/Spark clusters, and other enterprise applications.

- **R3**
  R3 instances are optimized for memory-intensive applications and offer lower price per GiB of RAM.

  R3 instances are recommended for high performance databases, distributed memory caches, in-memory analytics, genome assembly and analysis, Microsoft SharePoint, and other enterprise applications.

4. Accelerated Computing

- **P2**

  P2 instances are intended for general-purpose GPU compute applications.

  These are recommended for Machine learning, high performance databases, computational fluid dynamics, computational finance, seismic analysis, molecular modeling, genomics, rendering, and other server-side GPU compute workloads.

- **G2**
  G2 instances are optimized for graphics-intensive applications.
  These are recommended for 3D application streaming, video encoding, and other server-side graphics workloads.

- **F1**
  F1 instances offer customizable hardware acceleration with field programmable gate arrays (FPGAs).
  These are recommended for Genomics research, financial analytics, real-time video processing, big data search and analysis, and security.

5. Storage Optimized

- **I3 - High I/O**
  This family includes the High Storage Instances that provide Non-Volatile Memory Express (NVMe) SSD backed instance storage optimized for low latency, very high random I/O performance, high sequential read throughput and provide high IOPS at a low cost.
  These are recommended for NoSQL databases like Cassandra, MongoDB, Redis, in-memory databases such as Aerospike, scale out transactional databases, data warehousing, Elasticsearch, analytics workloads.

- **D2 – Dense Storage**

  D2 instances feature up to 48 TB of HDD-based local storage, deliver high disk throughput, and offer the lowest price per disk throughput performance on Amazon EC2.

  These are recommended for Massively Parallel Processing (MPP) data warehousing, MapReduce and Hadoop distributed computing, distributed file systems, network file systems, log, or data-processing applications

Instances Types Table

InstanceTypesAWS.xlsx

Instances Types LRZ (Mapping from AWS)

Instance Types LRZ						
Sr No.	**AWS Instance Type**	**LRZ Instance Types**	**vCPU**	**Memory (GiB)**	**Storage (GB)**	**Price**
1	t2.nano	LRZ.t2.nano	1	0.5	5	$0.0059 per Hour
2	t2.micro	LRZ.t2.micro	1	1	5	$0.012 per Hour
3	t2.small	LRZ.t2.small	1	2	5	$0.023 per Hour
4	t2.medium	LRZ.t2.medium	2	4	5	$0.047 per Hour
5	t2.large	LRZ.t2.large	2	8	5	$0.094 per Hour
6	t2.xlarge	LRZ.t2.xlarge	4	16	5	$0.188 per Hour
7	t2.2xlarge	LRZ.t2.2xlarge	8	32	5	$0.376 per Hour

- For the testing and research point we will use only two types of OS:
  1. Ubuntu 16.04 (x86)
  2. Ubuntu 16.04 (64 bit)
- As Amazon has different pricing models: On-Demand, Reserved Instances, Spot Instances and Dedicated Hosts pricing model [11]. But we are considering only **On-Demand** pricing model as On-Demand instances let you pay for compute capacity by the hour with no long-term commitments and it fits well for the testing and research point.
- Amazon also has ***Data Transfer OUT From Amazon EC2 To Internet*** but that is ***$0.0 for first 1GB/Month*** so we are currently neglecting it.

- Auto Scaling is enabled by Amazon CloudWatch and carries ***no additional fees***. Each instance launched by Auto Scaling is automatically enabled for monitoring and the **applicable *Amazon Cloudwatch charges*** will be applied. But those *charges* ***can also be neglected*** as You can get started with ***Amazon CloudWatch for free*** [12]. Many applications should be able to operate within these free tier limits.
    - New and existing customers also receive 3 dashboards of up to 50 metrics each per month at no additional charge
    - Basic Monitoring metrics (at five-minute frequency) for Amazon EC2 instances are free of charge, as are all metrics for Amazon EBS volumes, Elastic Load Balancers, and Amazon RDS DB instances.
    - New and existing customers also receive 10 metrics (applicable to Detailed Monitoring for Amazon EC2 instances, Custom Metrics, or CloudWatch Logs*), 10 alarms, and 1 million API requests each month at no additional charge.
    - New and existing customers receive extended retention of metrics free of charge.
    - New and existing customers also receive 5 GB of data ingestion and 5 GB of archived storage per month at no additional charge.

Autoscaling Decision Metrics (used for Autoscaling)

Metric	AWS	Kubernetes
CPU Utilization	Yes	Yes
Disk Reads	Yes	No
Disk Read Operations	Yes	No
Disk Writes	Yes	No
Disk Write Operations	Yes	No
Network In	Yes	No
Network Out	Yes	No
Other Custom		Still in Alpha mode but can be defined

Scaling Adjustment or Tuning Parameters

1. AWS

When a scaling policy is executed, it changes the current capacity of your Auto Scaling group using the scaling adjustment specified in the policy. A scaling adjustment can't change the capacity of the group above the maximum group size or below the minimum group size.

Auto Scaling supports the following adjustment types:

- **ChangeInCapacity**—Increase or decrease the current capacity of the group by the specified number of instances. A positive value increases the capacity and a negative adjustment value decreases the capacity.
  Example: If the current capacity of the group is 3 instances and the adjustment is 5, then when this policy is performed, Auto Scaling adds 5 instances to the group for a total of 8 instances.
- **ExactCapacity**—Change the current capacity of the group to the specified number of instances. Note that you must specify a positive value with this adjustment type.
  Example: If the current capacity of the group is 3 instances and the adjustment is 5, then when this policy is performed, Auto Scaling changes the capacity to 5 instances.
- **PercentChangeInCapacity**—Increment or decrement the current capacity of the group by the specified percentage. A positive value increases the capacity and a negative value decreases the capacity. If the resulting value is not an integer, Auto Scaling rounds it as follows:
  - Values greater than 1 are rounded down. For example, 12.7 is rounded to 12.
  - Values between 0 and 1 are rounded to 1. For example, .67 is rounded to 1.
  - Values between 0 and -1 are rounded to -1. For example, -.58 is rounded to -1.
  - Values less than -1 are rounded up. For example, -6.67 is rounded to -6.

Example: If the current capacity is 10 instances and the adjustment is 10 percent, then when this policy is performed, Auto Scaling adds 1 instance to the group for a total of 11 instances.

Here, you can also specify the minimum number of instances to scale (using the MinAdjustmentMagnitude parameter or Add instances in increments of at least in the console). For example, suppose that you create a policy that adds 25 percent and you specify a minimum increment of 2 instances. If you have an Auto Scaling group with 4 instances and the scaling policy is executed, 25 percent of 4 is 1 instance. However, because you specified a minimum increment of 2, Auto Scaling adds 2 instances.

2. Kubernetes

   With Horizontal Pod Autoscaling, Kubernetes automatically scales the number of pods in a replication controller, deployment or replica set based on observed CPU utilization (or, with alpha support, on some other, application-provided metrics).

3. Google Cloud Platform

   Its Autoscaling capabilities allows to automatically add or remove instances from a managed instance group based on increases or decreases in load.

4. Google Cloud Platform Cluster Autoscaler

   Cluster Autoscaler enables users to automatically resize clusters so that all scheduled pods have a place to run. If there are no resources in the cluster to schedule a recently created pod, a new node is added. On the other hand, if some node is underutilized and all pods running on it can be easily moved elsewhere then the node is deleted.

Autoscaling Policies

1. AWS

   The policy type determines how the scaling action is performed. AWS Auto Scaling supports the following policy types:

   - **Simple scaling**—Increase or decrease the current capacity of the group based on a single scaling adjustment.
   - **Step scaling**—Increase or decrease the current capacity of the group based on a set of scaling adjustments, known as step adjustments, that vary based on the size of the alarm breach.

When you create a step scaling policy, you add one or more step adjustments, which enables you to scale based on the size of the alarm breach. Each step adjustment specifies a lower bound for the metric value, an upper bound for the metric value, and the amount by which to scale, based on the scaling adjustment type.

For example, suppose that you have an alarm with a breach threshold of 50 and a scaling adjustment type of PercentChangeInCapacity. You also have scale out and scale in policies with the following step adjustments:

Scale out policy			
Lower bound	**Upper bound**	**Adjustment**	**Metric value**
0	10	0	50 <= value < 60
10	20	10	60 <= value < 70
20	null	30	70 <= value < +infinity
Scale in policy			
Lower bound	**Upper bound**	**Adjustment**	**Metric value**
-10	0	0	40 < value <= 50
-20	-10	-10	30 < value <= 40
null	-20	-30	-infinity < value <= 30

Your group has both a current capacity and a desired capacity of 10 instances. The group maintains its current and desired capacity while the aggregated metric value is greater than 40 and less than 60.

If the metric value gets to 60, Auto Scaling increases the desired capacity of the group by 1 instance, to 11 instances, based on the second step adjustment of the scale-out policy (add 10 percent of 10 instances). After the new instance is running and its specified warm-up time has expired, Auto Scaling increases the current capacity of the group to 11 instances. If the metric value rises to 70 even after this increase in capacity, Auto Scaling increases the desired capacity of the

group by another 3 instances, to 14 instances, based on the third step adjustment of the scale-out policy (add 30 percent of 11 instances, 3.3 instances, rounded down to 3 instances).

If the metric value gets to 40, Auto Scaling decreases the desired capacity of the group by 1 instance, to 13 instances, based on the second step adjustment of the scale-in policy (remove 10 percent of 14 instances, 1.4 instances, rounded down to 1 instance). If the metric value falls to 30 even after this decrease in capacity, Auto Scaling decreases the desired capacity of the group by another 3 instances, to 10 instances, based on the third step adjustment of the scale-in policy (remove 30 percent of 13 instances, 3.9 instances, rounded down to 3 instances).

2. Google Cloud Platform

To create an Autoscaler, you must specify the Autoscaling policy and a target utilization level that the Autoscaler uses to determine when to scale the group. You can choose to scale using the following policies:

- Average CPU utilization
- Stackdriver Monitoring metrics
- HTTP load balancing serving capacity, which can be based on either utilization or requests per second.
- Google Cloud Pub/Sub queuing workload (Alpha)

The Autoscaler will collect information based on the policy, compare it to your desired target utilization, and determine if it needs to perform scaling.

The target utilization level is the level at which you want to maintain your virtual machine instances. For example, if you scale based on CPU utilization, you can set your target utilization level at 75% and the Autoscaler will maintain the CPU utilization of the specified group of instances at or close to 75%. The utilization level for each metric is interpreted differently based on the Autoscaling policy.

References

[1] "Auto Scaling," Amazon Web Services, [Online]. Available: http://aws.amazon.com/autoscaling/. [Accessed 11 May 2017].

[2] "Autoscaling guidance," Microsoft Azure, [Online]. Available: https://github.com/Azure/azure-content-nlnl/blob/master/articles/best-practices-auto-scaling.md. [Accessed 2017 May 11].

[3] "Docker Engine API and SDKs," Docker, [Online]. Available: https://docs.docker.com/engine/api/. [Accessed 11 May 2017].

[4] H. Jesheen, "Auto Scaling with Docker," 13 July 2016. [Online]. Available: https://botleg.com/stories/auto-scaling-with-docker/. [Accessed 11 May 2017].

[5] "What Is Auto Scaling?," Amazon Web Services, [Online]. Available: http://docs.aws.amazon.com/autoscaling/latest/userguide/WhatIsAutoScaling.html. [Accessed 11 May 2017].

[6] "Overview of autoscale in Microsoft Azure Virtual Machines, Cloud Services, and Web Apps," Microsoft Azure, 2 March 2016. [Online]. Available: https://docs.microsoft.com/en-gb/azure/monitoring-and-diagnostics/monitoring-overview-autoscale. [Accessed 12 May 2017].

[7] "Horizontal Pod Autoscaling," Kubernetes, [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. [Accessed 12 May 2017].

[8] "Autoscaling Groups of Instances," Google Cloud Platform, [Online]. Available: https://cloud.google.com/compute/docs/autoscaler/. [Accessed 13 May 2017].

[9] "Cluster Autoscaler," Google Cloud Platform, [Online]. Available: https://cloud.google.com/container-engine/docs/cluster-autoscaler. [Accessed 13 May 2017].

[10] "Amazon EC2 Instance Types," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ec2/instance-types/. [Accessed 13 May 2017].

[11] "Amazon EC2 Pricing," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ec2/pricing/. [Accessed 14 May 2017].

[12] "Amazon CloudWatch Pricing," Amazon Web Services, [Online]. Available: https://aws.amazon.com/cloudwatch/pricing/. [Accessed 14 May 2017].