Event Driven Scaling

Enable Scalability

Best Practice

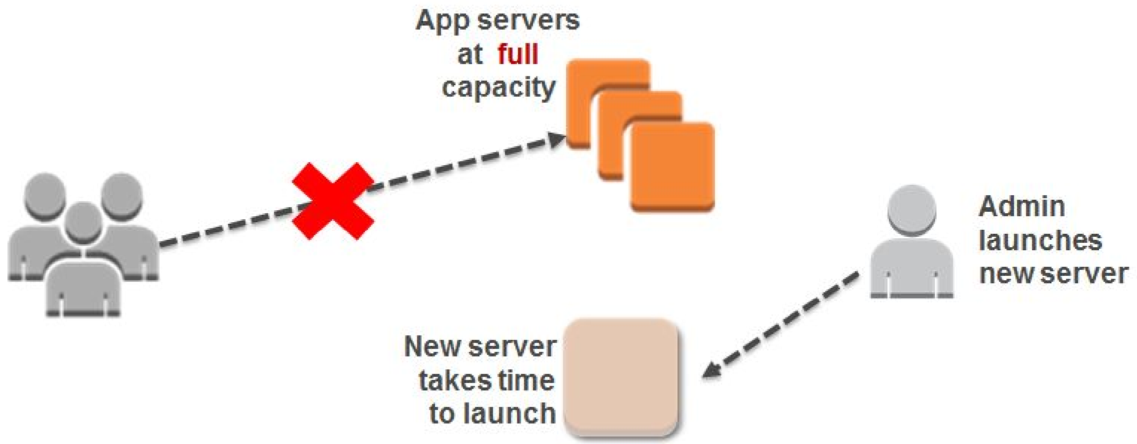
Best Practice

Ensure that your architecture can handle changes in demand

A key advantage of a cloud-based infrastructure is how quickly you can respond to changes in resource needs.

Scalability of Infrastructure

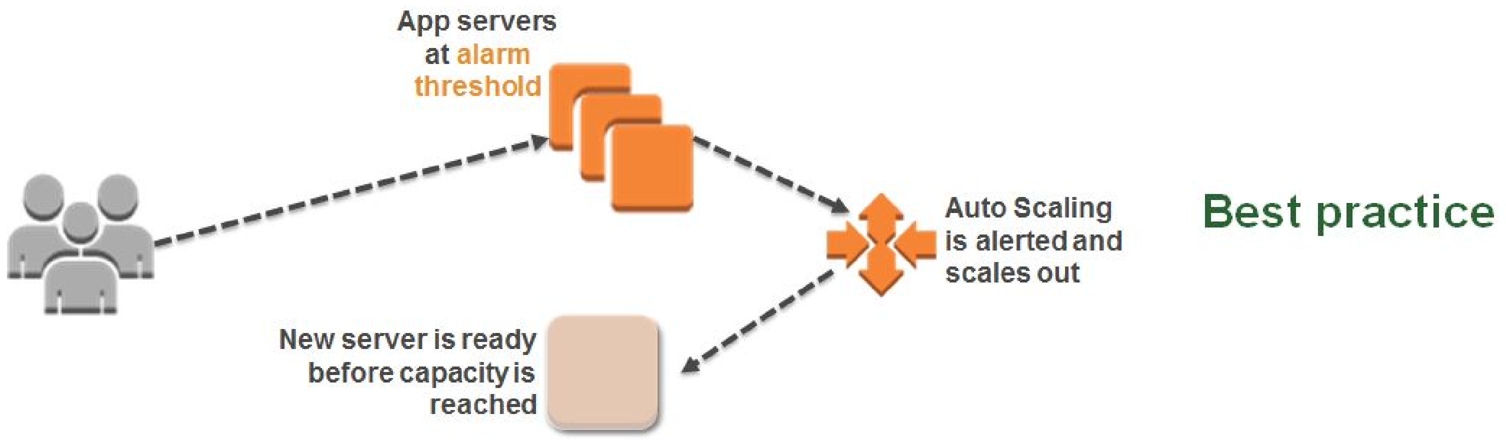
Anti-patterns


App servers at full capacity
Admin launches new server
New server takes time to launch

Anti-pattern

Best Practice

Event driven



Vertical vs. Horizontal Scaling

- Vertical scaling keeps adding memory and CPUs as the workload increases.
- Eventually, this will hit the limit. Also, there are some performance concerns.
- If you are running Java applications, more memory (more Java heap) means it takes longer for garbage collection to run.

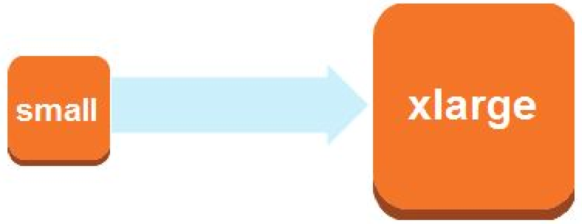
Vertical vs. Horizontal Scaling

- It may introduce longer pause time.
- Another point to consider is that vertical scaling may require the server to be re-booted.
- Horizontal scaling is virtually limitless.
- Horizontal scaling is the ideal solution to handle growing workload
- If you're going to rely on horizontal scaling, your application be designed to fully take advantage of it.

Vertical vs. Horizontal Scaling

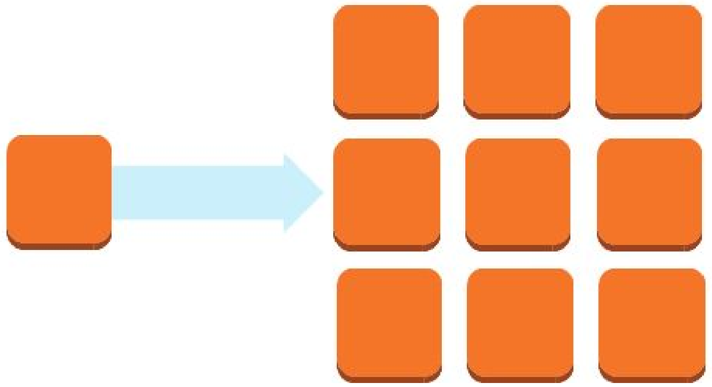
Vertical scaling
Scale up and down

Change in the specifications of instances (more CPU, memory, etc.)



Horizontal scaling
Scale in and out

Change in the number of instances (Add and remove instances as needed)



Instrumenting

Instrumenting your architecture

Capture information

To leverage auto-scale in an efficient way, you need insight into your resources, e.g.:

How much of your infrastructure is actually being used?

Is your application's performance or availability being affected by a lack of sufficient capacity

Without Information

You are flying blind without information:

Monitoring

Your architecture should:

Monitor components in your infrastructure based on what you identify.

Send notifications and triggers auto scaling actions based on metrics you specify.

Gathering Statistics

Azure and AWS offer:

A distributed statistics gathering system; collect and track metrics.

Metrics are collected at the hypervisor level by default**

**what might these be?

Custom Metrics

Custom metrics are also desirable:

The ability to create and use custom metrics of data generated by your own applications and services.

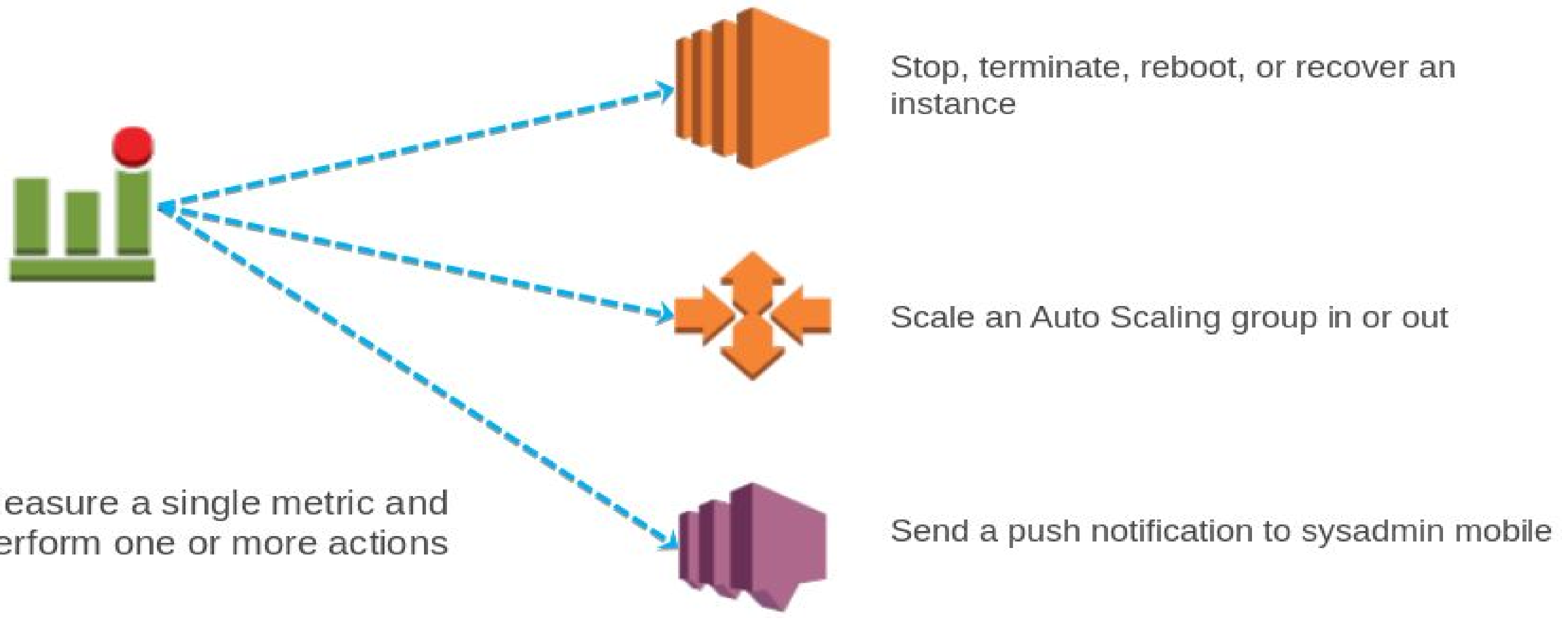
Q: Can you think of any custom metrics for recent architectures you've been involved with?

Example Statistics

Database: If number of simultaneous connections is > 10 for one minute…

Compute Node: If CPU utilization is > 60% for 5 minutes…

Alarms and Actions


Stop, terminate, reboot, or recover an instance
Scale an Auto Scaling group in or out
Send a push notification to sysadmin mobile

Measure a single metric and perform one or more actions

Monitoring Problems

Monitoring is necessary in systems operation.

However, because the monitoring service in the cloud cannot monitor the internal workings of a virtual server (the operating system, middleware, applications, and so forth), you need to have an independent monitoring system.

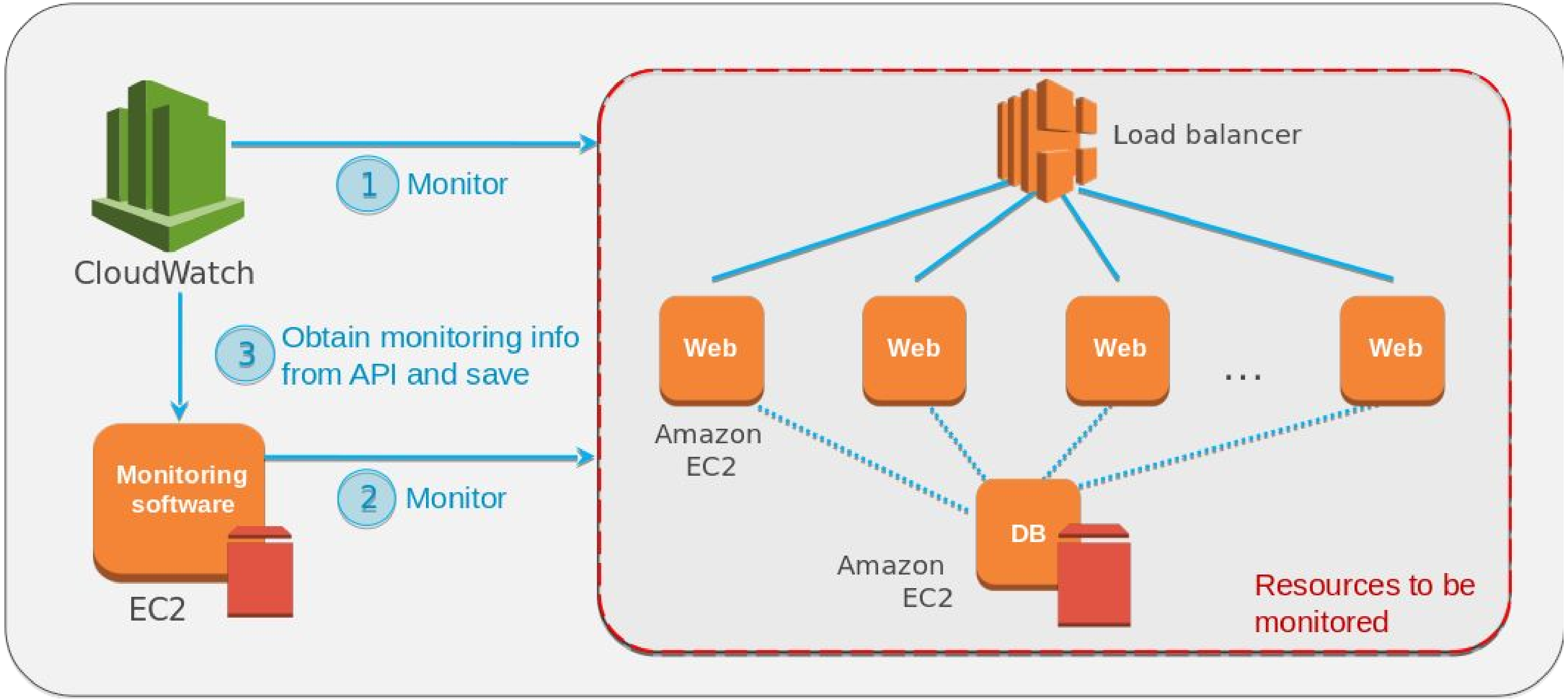
Use of Plug-ins

Install monitoring software (Nagios, Zabbix, Munin, or the like).

Use a plug-in to obtain monitoring information using the cloud vendor and to write that information to the monitoring software.

Use the plug-in to perform monitoring, including the information from AWS.

In AWS


CloudWatch
1 Monitor
Load balancer
3 Obtain monitoring info from API and save
Web
Web
Web
...
Web
Amazon EC2
Monitoring software
EC2
2 Monitor
DB
Amazon EC2
Resources to be monitored

In Azure

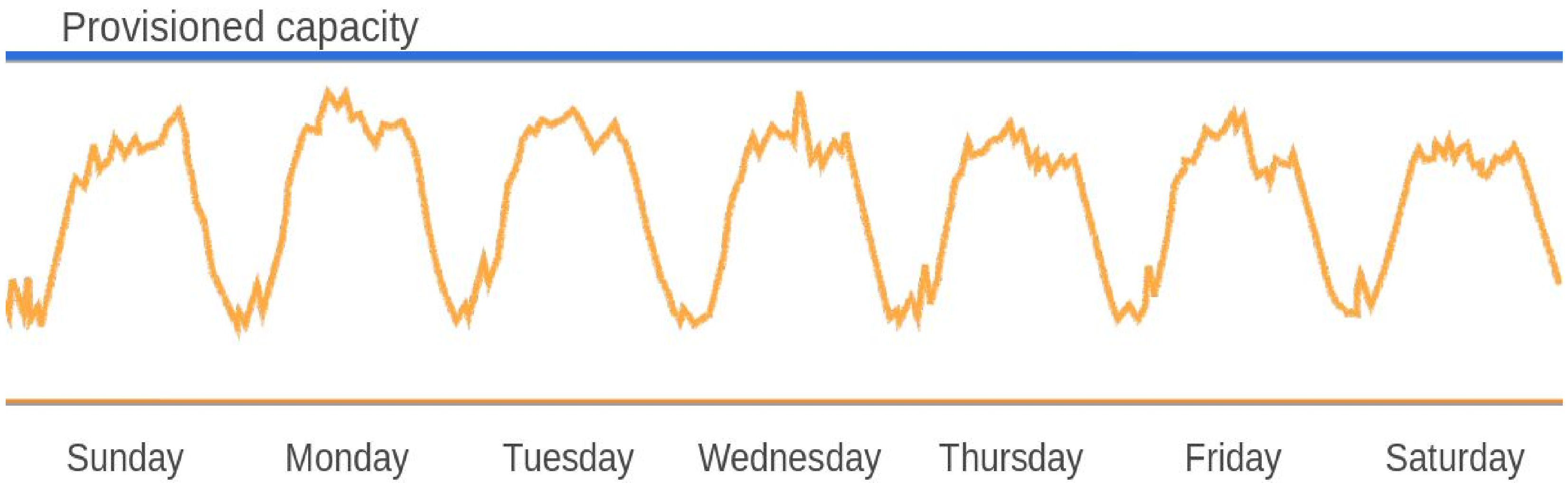
Azure provides a similar service for VM statistics.

The collection of custom metrics seems possible using Microsoft Azure Monitor REST API Reference (Java, C#, Javascript bindings) (https://msdn.microsoft.com/en-us/library/azure/dn931943.aspx)

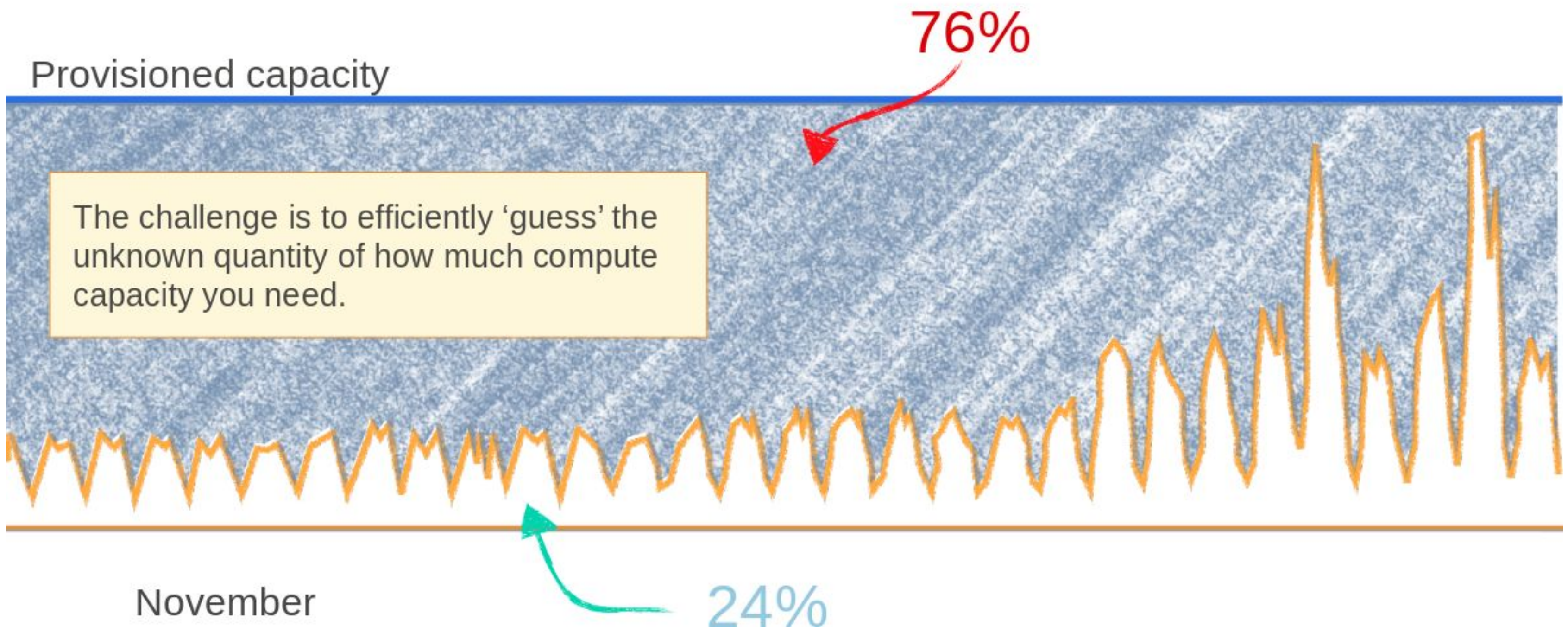
Auto Scaling Compute

Scaling your compute service on demand

Period Traffic Patterns



Variance in Demand



The challenge is to efficiently 'guess' the unknown quantity of how much compute capacity you need.

Underutilization

- The problem here that about 76% of the resources are idle for most of the year.
- But if you don’t have enough compute capability to support the seasonal peak, the server can crash and your business can lose customer confidence.

Load Balancing

The foundation of a scalable tier includes the use of Load balancers in the architecture.

These load balancers not only send traffic to instances, but should also send metrics to a monitoring service.

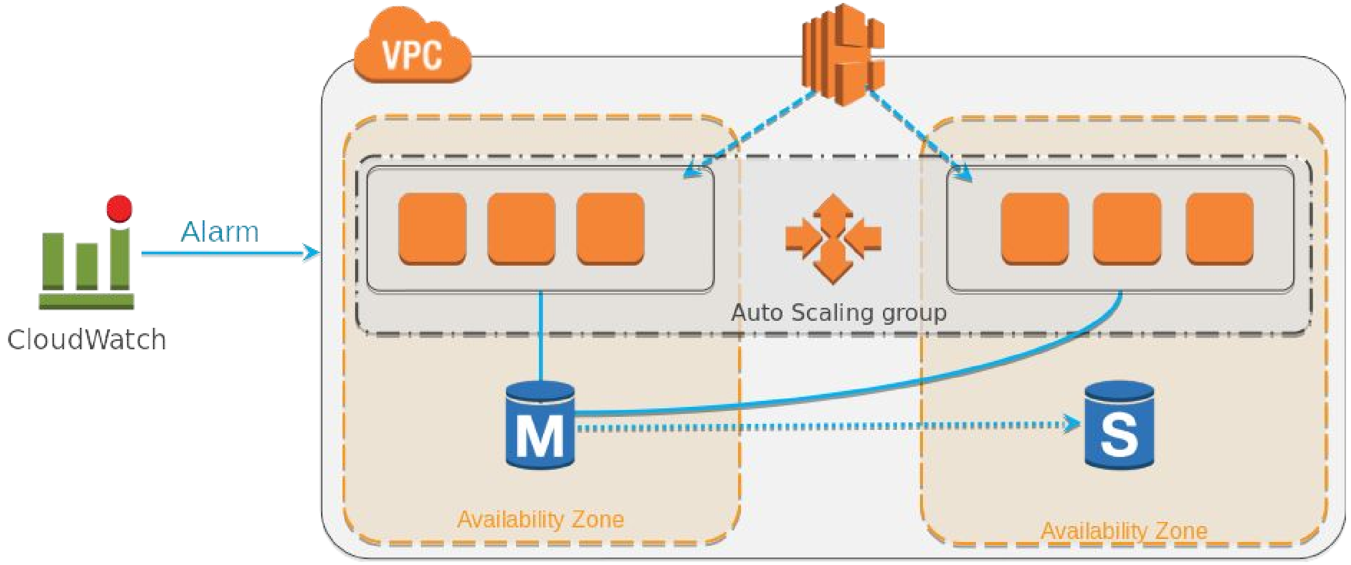
Load Balancer Metrics

Typically your load balancer should monitor:

- high request latency
- over utilized
- servers not responding

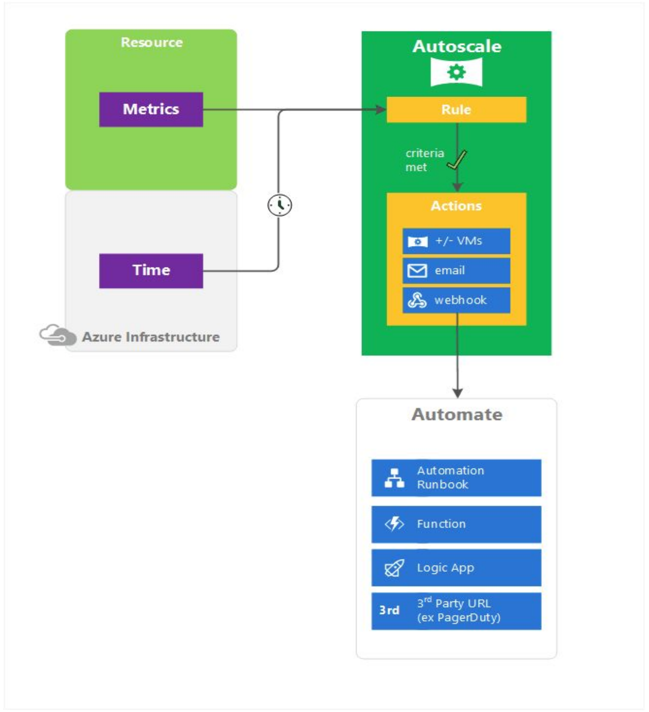
Resource needs

AWS Architecture



Azure autoscale

Azure approach is conceptually the same as AWS, but with some differences. The role of the load balancer is not that clear



Scale out, and in

- Autoscale allows you to have the right amount of resources running to handle the load on your application.
- It allows you to add resources to handle increases in load and also save money by removing resources that are sitting idle.

How Does Auto Scaling Work?

Generally, the following functions are present in auto scale archiectures

- Launch configuration
- Auto Scaling group
- Auto Scaling policy *or*
- Scheduled policy

Launch Configurations

In IaaS architectures, you specify

- OS Config (Linux, Win, DB, HTTP Server etc)
- Instance type (VM spec)
- User data (Req config files)
- Security (Permissions for the server)

Auto Scaling Group

In IaaS architectures, you specify

- Min and Max
- Zone or subnet
- Load balancer to control the group
- Desired capacity

Auto Scaling Policy

Specifies when to increase or decrease instances based on alarms or metrics collected.

Auto Scaling Schedule

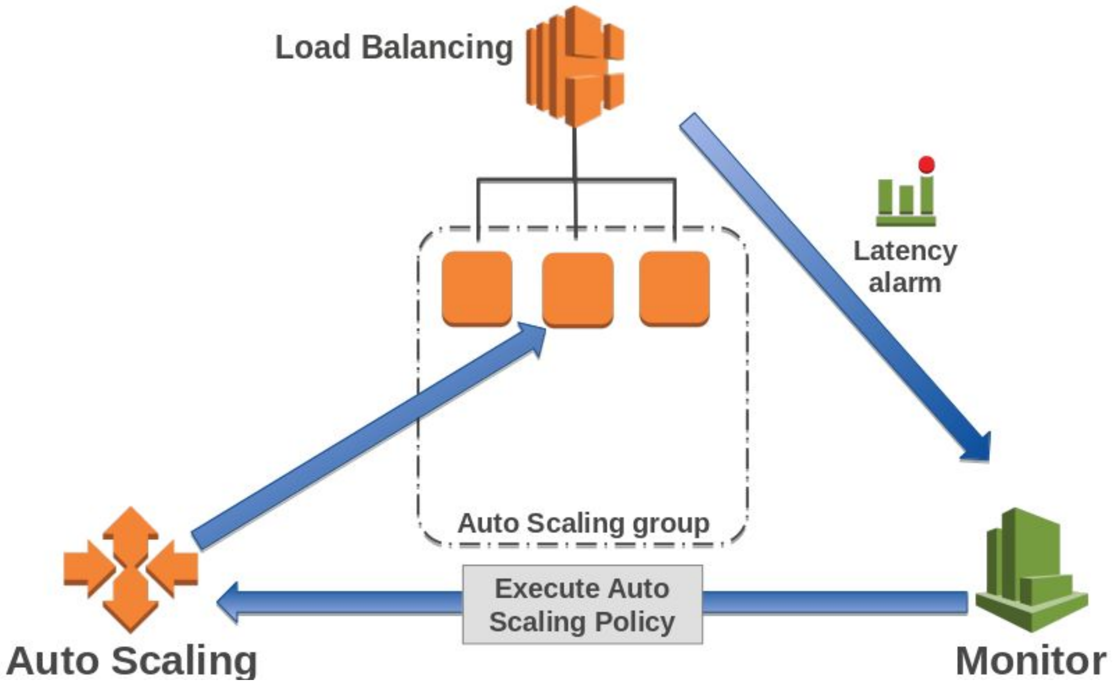
Some cases you know with certainty the demand schedule, and you can design your scaling schedule with this in mind

Q: In what situations might you know the demand schedule with certainty?

Auto Scale vs Schedule

Auto Scaling Policy	Scheduled Action
• Manually initiate a scaling activity • Dynamic scaling with CloudWatch to initiate a scaling activity based on a metric	• Prior knowledge so you can be proactive (predictable schedule)
Parameters example: • AutoScalingGroupName • HonorCooldown • PolicyName	Parameters example: • ScheduledActionName • AutoScalingGroupName • DesiredCapacity • StartTime
Usage scenario: If CPU utilization reaches 70%, add an instance. In contrast, when the CPU utilization is low, remove an instance to reduce the cost.	**Usage scenario:** Traffic to your web application starts to increase on Wednesday, remains high on Thursday, and starts to decrease on Friday.

Load Balancing, Monitor, And Auto Scaling



Minimum Capacity Size

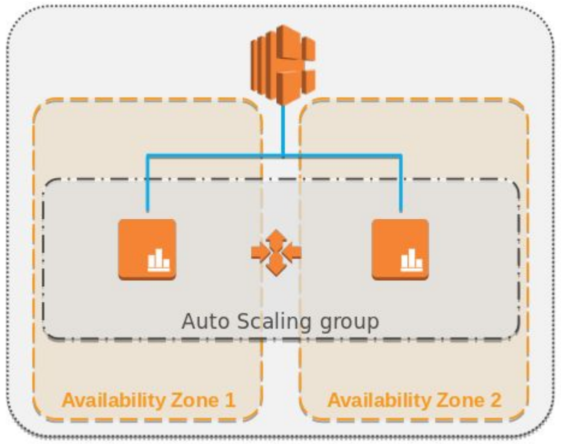
- Desired capacity
- Minimum capacity
- Maximum capacity
- What would be a good minimum capacity to set it to?
- What would be a good maximum capacity to set it to?

HA and Auto scaling

In a HA Context:

If you require at least one instance per AZ to start with, the minimum capacity size should be set to the number of AZs.**

known as a ***region pair* in Azure



Minimum = two instances (# of AZs)

Desired capacity = two instances (Min.)

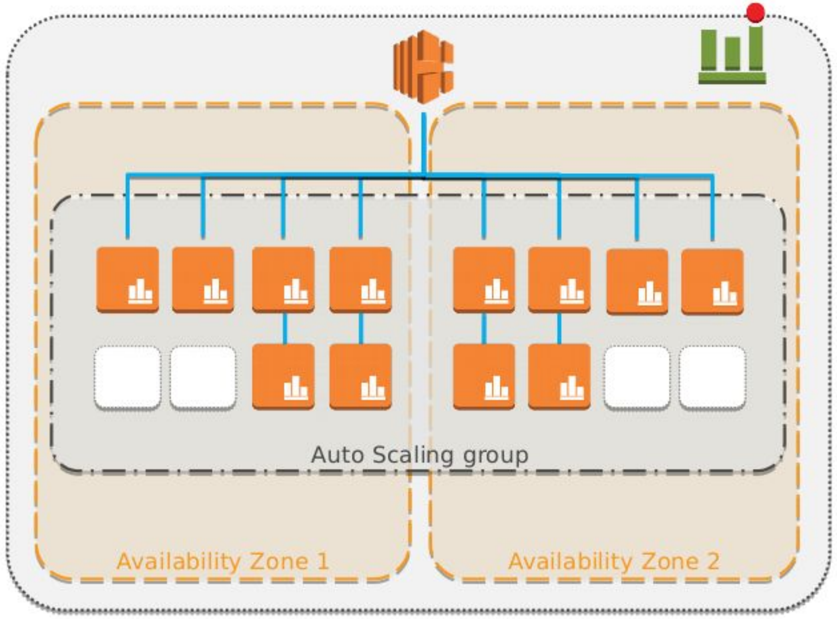
Capacity Rubrics

Deciding on the minimum capacity size depends on the type of applications that you are running

- If you are processing batches that run once a week, you might want to set the minimum to zero.
- Remember that it takes minutes to launch a new instance (depending on the complexity of your launch configuration).
- You may not be able to afford zero minimum capacity to start with.

Scaling Scenario

CPU utilization triggers the alarm: capacity is doubled until CPU utilization drops below 60% or max capacity is reached.



Scenario:

Auto Scaling group

- Minimum = 2
- Maximum = **12**

Auto Scaling policy

- When CPU utilization is greater than 60%
- Add 100% of group = **double the capacity**

Example

Rules

Scale Out	**Scale In**
Add 2 instances when average CPU is 80-100%	Remove 1 instance when average CPU is 20-40%
Add 1 instance when average CPU is 60-80%	Remove 2 instances when average CPU is 0-20%

Limits - Minimum: 5 instances & Maximum: 20 instances

Worked Example

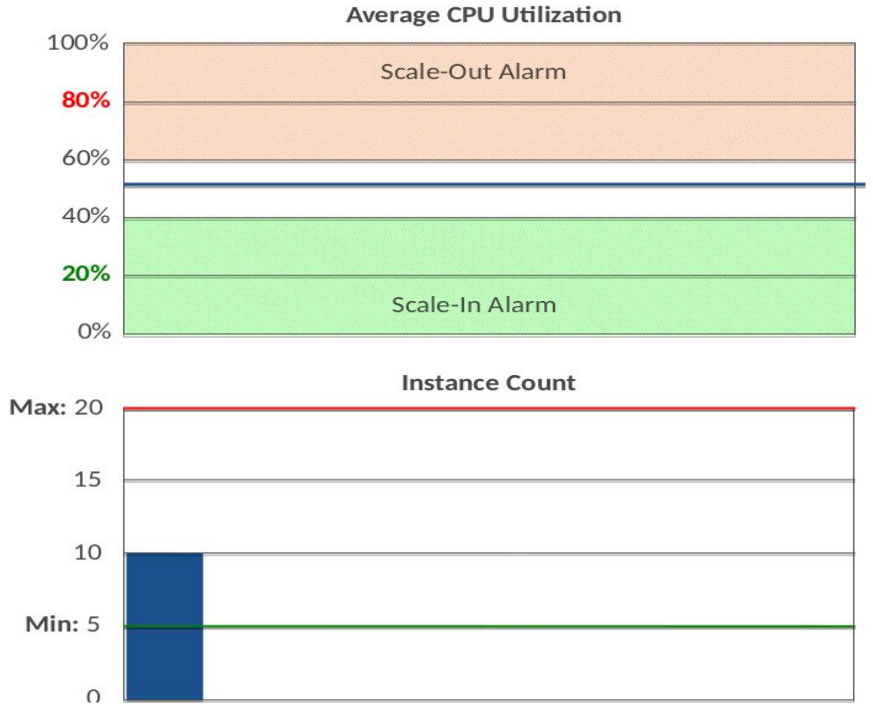
Auto Scaling Steps

Alarms:

- Add 2 instances when average CPU is 80-100%
- Add 1 instance when average CPU is 60-80%
- Remove 1 instance when average CPU is 20-40%
- Remove 2 instances when average CPU is 0-20%

Limits:

Minimum:	5 instances
Maximum:	20 instances



Worked Example

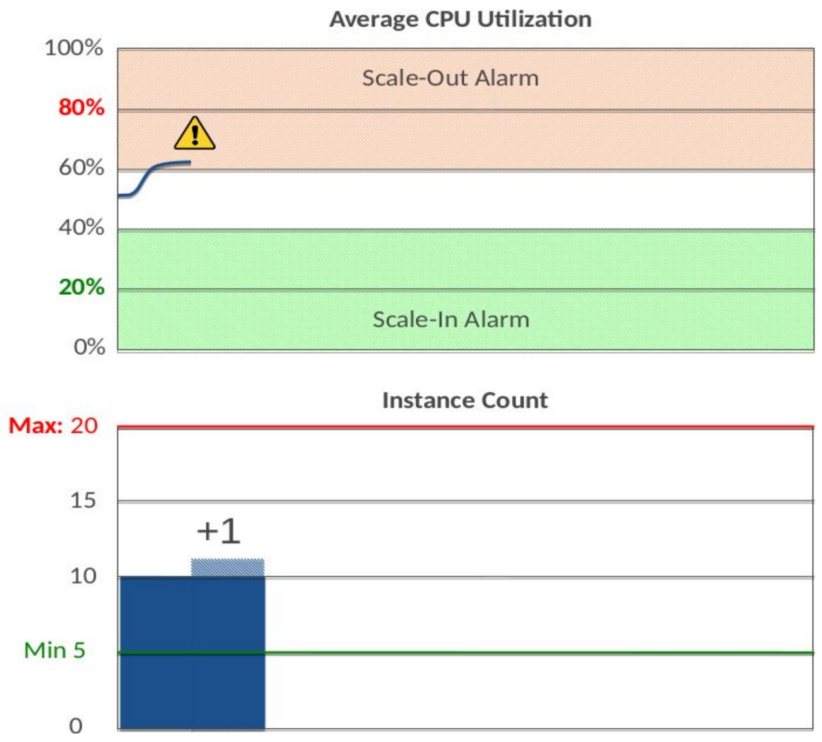
Auto Scaling Steps

As usage increases:

- CPU utilization goes up

When CPU utilization is 60-80%:

- Scale-out alarm is triggered
- Add 1 step policy is applied
- New instance is launched but not added to the aggregated group metrics until after warm up period expires



Worked Example

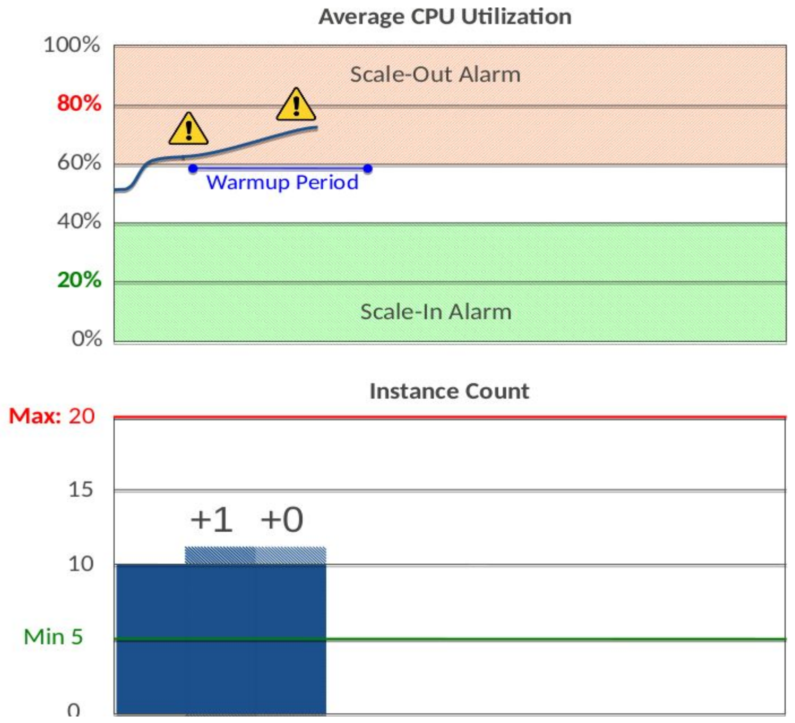
Auto Scaling Steps

As usage increases:

- CPU utilization goes up.

While waiting for new instance:

- CPU utilization remains high.
- Another alarm period is triggered.
- Since current capacity is still 10 during the warmup period, and desired capacity is already 11, no additional instances are launched.



Worked Example

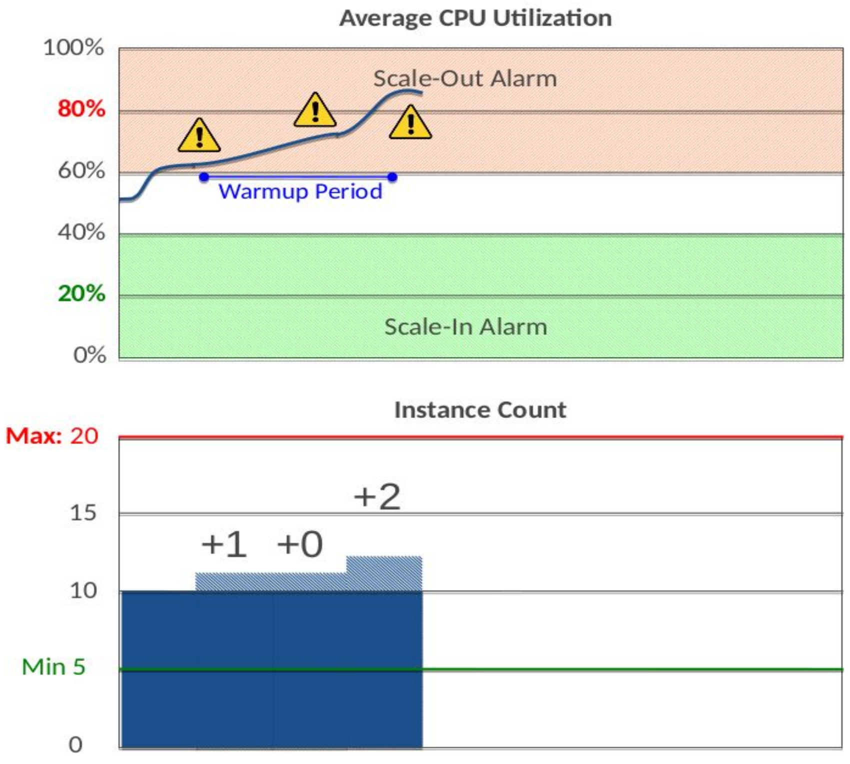
Auto Scaling Steps

As usage increases further:

- CPU utilization goes up

When CPU utilization is 80-100%:

- Scale-out alarm is triggered
- Add 2 step policy is applied
- Since the alarm occurred during a warm up period, two instances are launched less the one instance added during the first alarm
- Again new instances are not added to aggregated group metrics



Worked Example

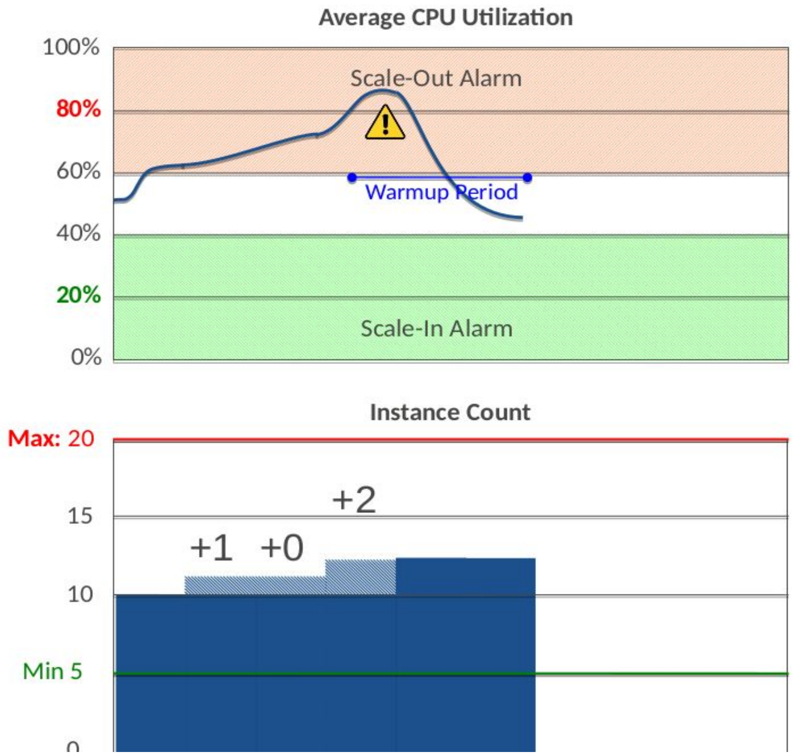
Auto Scaling Steps

As capacity matches usage:

- CPU utilization stabilizes

When CPU utilization is 40-60%:

- No alarms are triggered
- After warm up periods expire, new instances are added to the aggregated group metrics



Worked Example

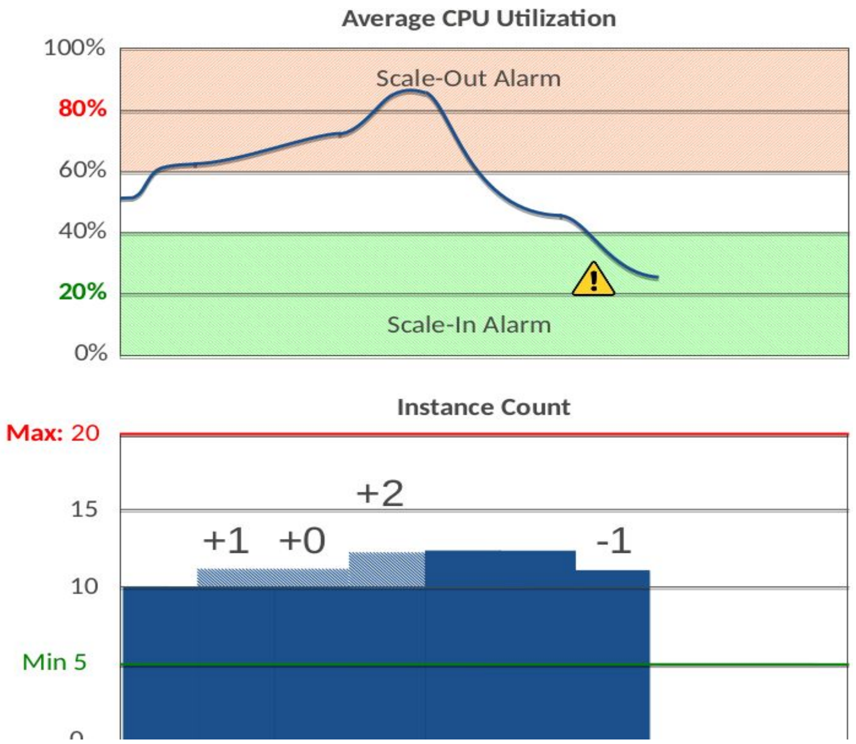
Auto Scaling Steps

As usage decreases:

- CPU utilization goes down

When CPU utilization is 20-40%:

- Scale-in alarm is triggered
- Remove 1 step policy is applied
- An instance is removed from the Auto Scaling group and from the aggregated group metrics



Worked Example

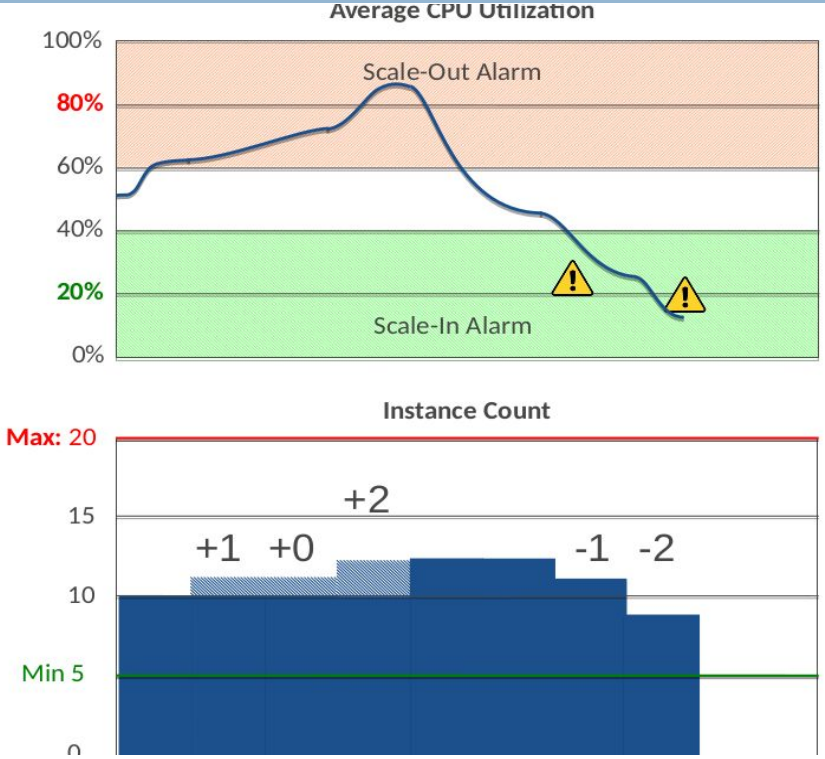
Auto Scaling Steps

As usage decreases:

- CPU utilization goes down further

When CPU utilization is 0-20%:

- Scale in alarm is triggered
- Remove 2 step policy is applied
- Two instances are removed from the Auto Scaling group and from the aggregated group metrics



Worked Example

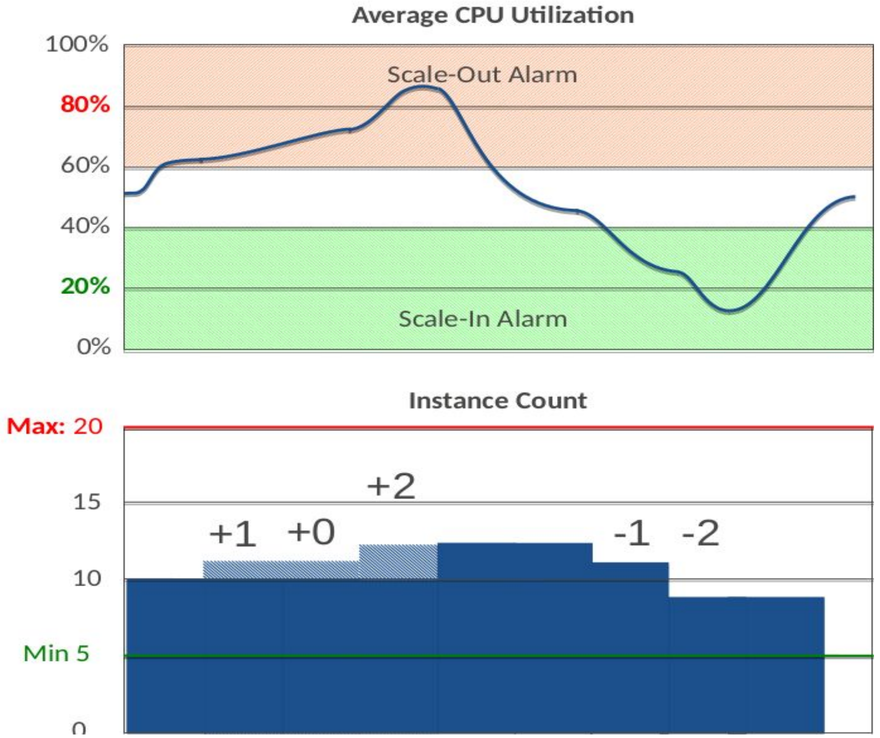
Auto Scaling Steps

As capacity matches usage:

- CPU utilization stabilizes

When 40% < CPU Utilization < 60%

- No alarm is triggered
- No step policies are applied
- No instances are added or removed from service



Best Practice

- Avoid Auto Scaling thrashing.
- Be more cautious about scaling in; avoid aggressive instance termination.
- Scale out early, scale in slowly.
- Set the min and max capacity parameter values carefully.

Best Practice

- Use lifecycle hooks.
- Perform custom actions as Auto Scaling launches or terminates instances.
- Stateful applications will require additional automatic configuration of instances launched into Auto Scaling groups.

Finally

Finally, remember: Instances can take several minutes after launch to be fully usable.

Scaling Storage

Scale up and scale out

Scaling Storage

Database servers sometimes need resizing

In a public cloud there are two ways:

- Unmanaged - you scale the DB manually
- Managed - use a service such as AWS RDS or Azure Cloud SQL to easily resize

Read Replicas and Caches

Offload read traffic to Read Replicas

For increased performance, put a cache in front of your DB such as: **Memcached or Redis**

AWS have a managed cache: **ElastiCache**

Sharding

Without shards, all data resides in one partition.

Example: Users by last name, A to Z, in one database.

With sharding, split your data into large chunks (shards).

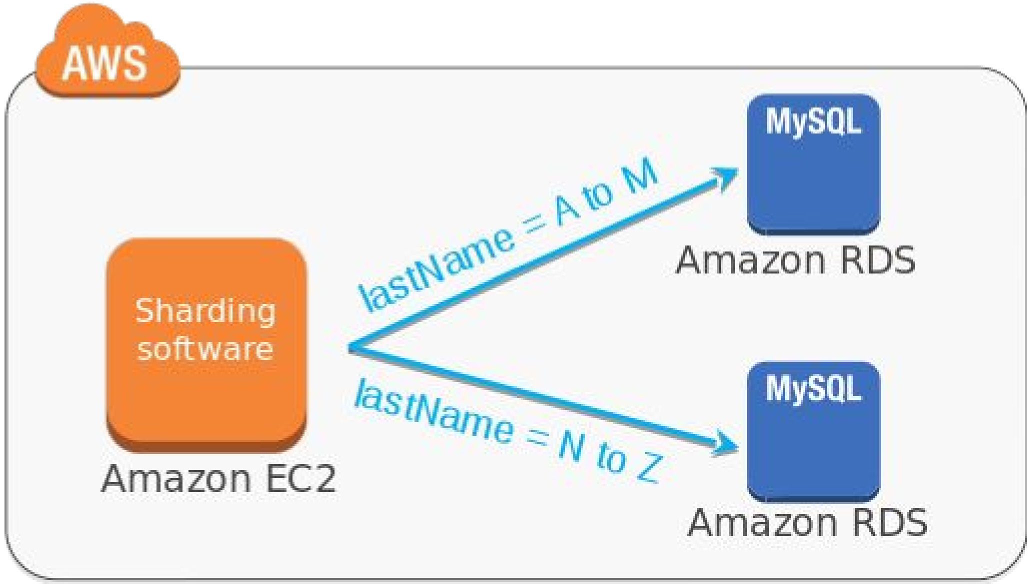
Sharding

Example: Users by last name, A through M, in one database; N through Z in another database.

In many circumstances, sharding gives you higher performance and better operating efficiency.

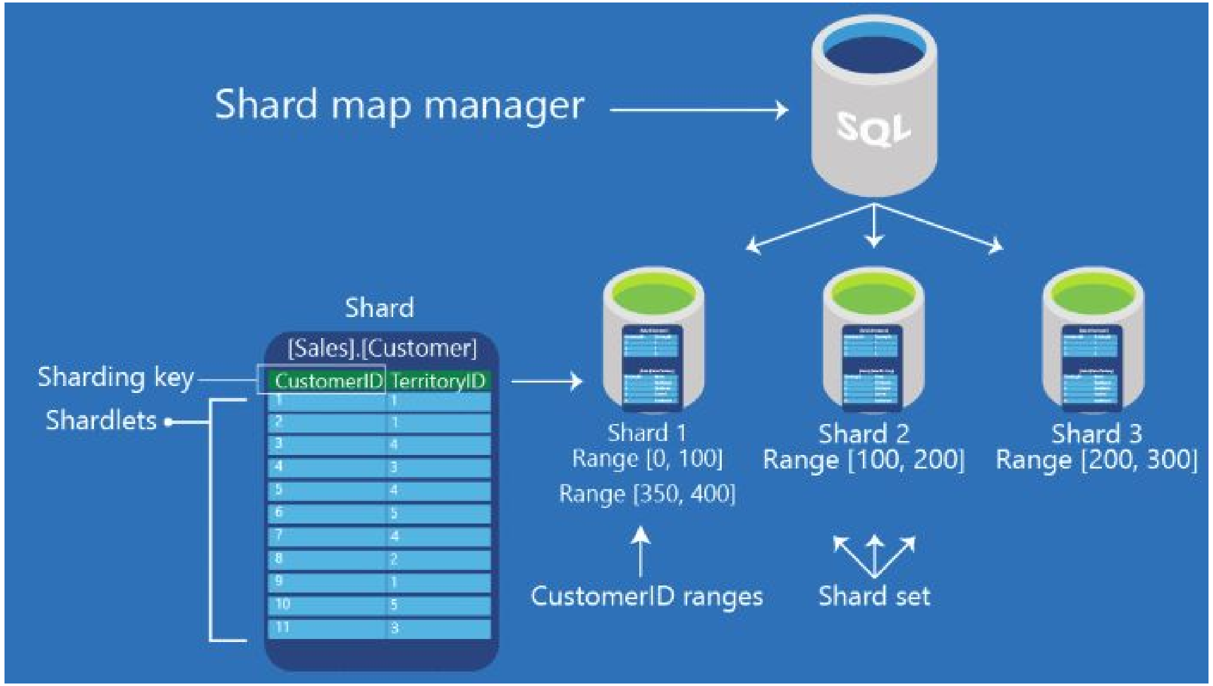
Sharding

AWS



Sharding

Azure



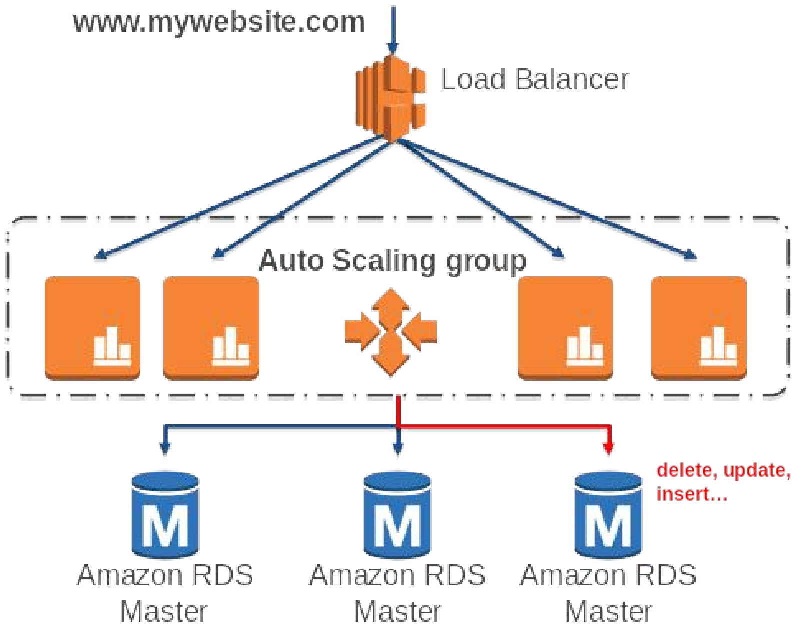
CustomerID	TerritoryID
1	1
2	1
3	4
4	3
5	4
6	5
7	4
8	2
9	1
10	5
11	3

Read Replicas

- Horizontally scale for read-heavy workloads
- Offload reporting
- Keep in mind…
- Replication is asynchronous
- [AWS] Currently available for Amazon Aurora, MySQL, and PostgreSQL (9.3.5 and later)

Read Replicas

AWS



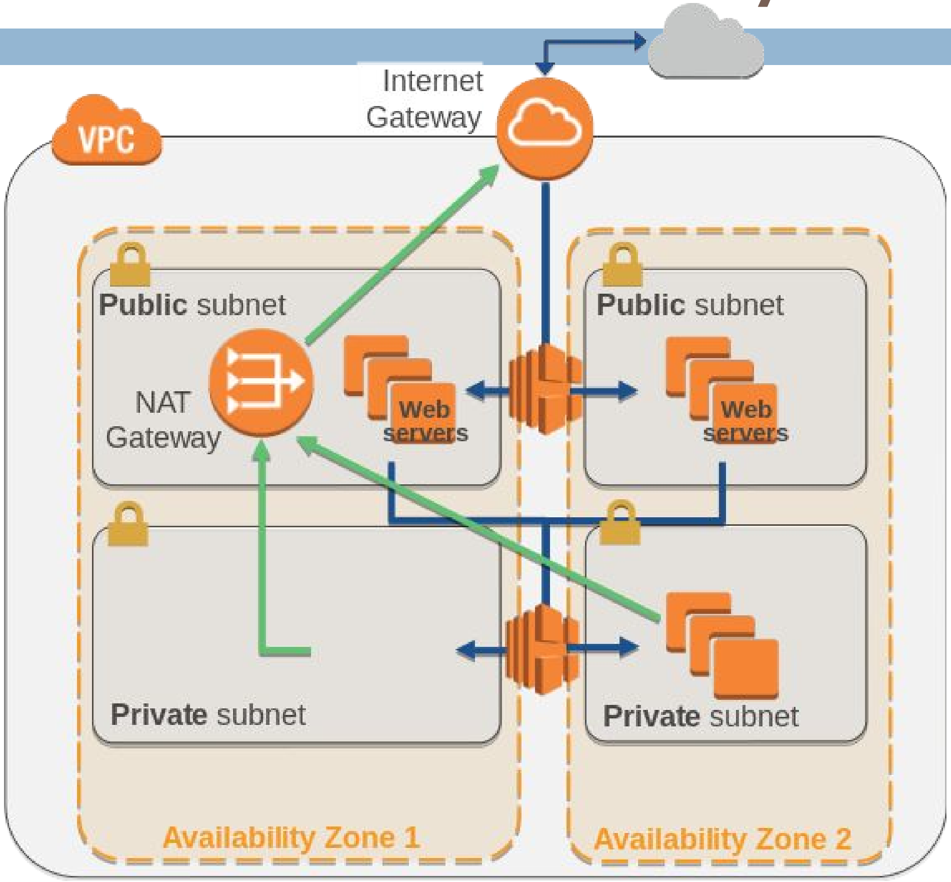
Azure Database Options

Readable secondary databases: An application can access a secondary database for read-only operations using the same or different security principals used for accessing the primary database.

The secondary databases operate in snapshot isolation mode to ensure replication of the updates of the primary (log replay) is not delayed by queries executed on the secondary

Increase the Availability Here?

Consider


Internet
Gateway
VPC
Public subnet
NAT
Gateway
Web
servers
Public subnet
Web
servers
Private subnet
Private subnet
Availability Zone 1
Availability Zone 2