The Art of Auto Scaling:

Learn the Latest Best Practices and Avoid Common Mistakes

With Sample Architecture Diagrams and CloudFormation Templates

© Logicworks. All rights reserved.

Introduction

Auto Scaling has long been a major selling point of cloud computing. But like most popularized technology features, it has accumulated its fair share of misconceptions. These common mistakes tend to get in the way of constructive conversations about cloud architecture, and usually mislead IT leaders into believing that Auto Scaling is simple, quick to set up, and always ensures 100% uptime.

IaaS platforms make Auto Scaling possible, usually in a way that is much more straightforward than scaling up in a datacenter. But if you visit Amazon Web Services (AWS) and spin up an instance, you will quickly discover that public cloud does not "come with" Auto Scaling.

Over the last decade, Logicworks has been continually refining their Auto-Scaling solutions, helping hundreds of customers on AWS, from startups to large enterprises leverage Auto Scaling to enable growth, high availability, and resilience. We have consolidated our top tips into this eBook to provide you with a blueprint for designing a resilient and scalable system while also avoiding those pesky misconceptions and mistakes.

Table of Contents

Part 1: Foundations of Auto Scaling

- Key Terms
- Why Auto Scaling?
- Real-Life Example of Auto Scaling

Part 2: The Auto Scaling Process

Part 3: Complexities of Auto Scaling

- Instance Bootstrapping
- Windows
- Spiky Workloads

© Logicworks. All rights reserved.

Part 1: Foundations of Auto Scaling

Auto Scaling is a feature of several AWS services that allows you to automatically add or remove capacity according to pre-set conditions. Although Auto Scaling is usually associated with Amazon EC2 instances, you can also use Auto Scaling with Amazon ECS tasks, Dynamo DB tables, Amazon Aurora replicas, and more.

Why Auto Scaling?

Scalability. Let your infrastructure grow with your requirements with dynamic load-based scaling, rather than over provisioning to meet peak demand.

Self-healing. To improve resiliency, put instances into a fixed-size Auto Scaling Group. If an instance fails, it is automatically replaced. The simplest use case is an Auto Scaling Group has a minimum size of 1 and a max of 1.

Cost savings. Whether you're using Auto Scaling Groups for resiliency or scalability, you're paying for what you need rather than paying for overprovisioned or redundant capacity.

Key Terms

Amazon Machine Image (AMI): A template that contains all the information required to launch an instance.

Launch Template: The template and other information used to build an instance in an Auto Scaling Group. Usually in the form of an AMI ID.

User-data: Data that you pass to configure an instance at or after launch. Either a shell script or cloud-init directives. There's a user-data field in Auto Scaling Group launch configurations for you to copy/paste your shell script. Usually used to do things like install packages or agents after instance launch.

Auto Scaling Groups: The logical grouping of identical instances launched by the same launch configuration with its corresponding minimum, maximum, and desired number.

Scale-out event: Any event that triggers the launch of one or more instances to the Auto Scaling group.

Scale-in event: Any event that triggers the termination of one more instance in the Auto Scaling Group.

Cooldown period: A period you can set after a scaling event occurs, so that you can wait for the effects of a scale-out or scale-in to occur before any further events are triggered.

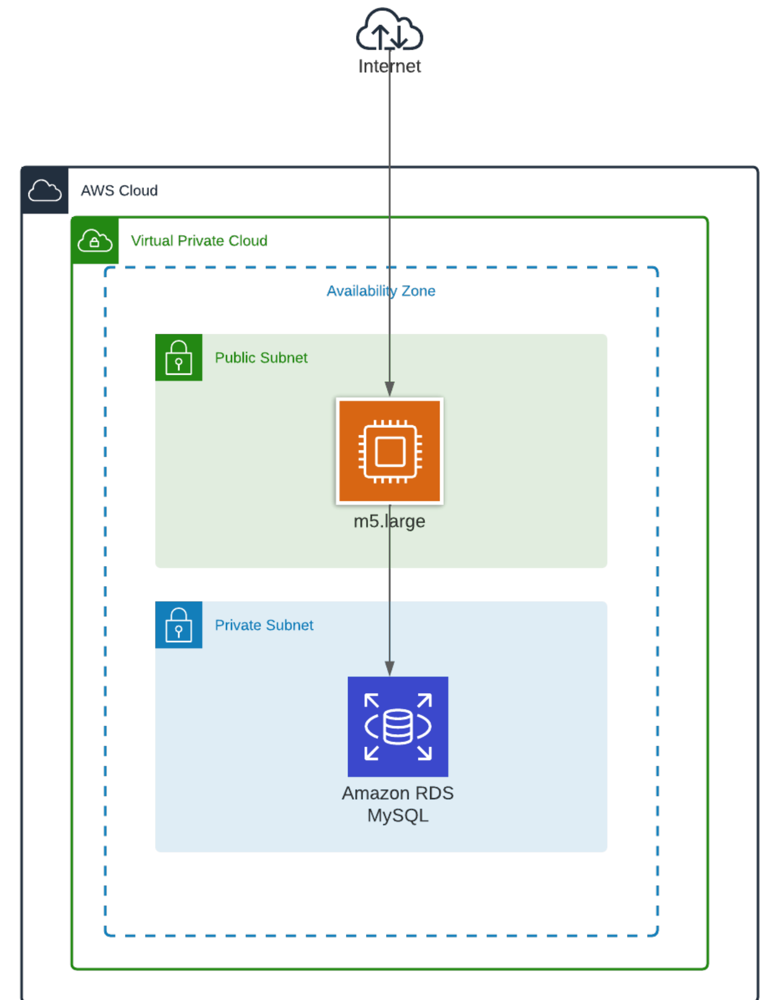
© Logicworks. All rights reserved.

Example: Simple E-Commerce Website

The basics of Auto Scaling are easiest to see in an example. Let's say you have an application and infrastructure with the following conditions:

- Basic 2-tier web application
- Web: Single Amazon EC2 instance (m5.large) running Nginx in public subnet
- DB: Single Amazon RDS MySQL database in private subnet
- 10,000 average daily visitors

Now let's say that during a transaction, you get an Amazon CloudWatch alarm that your Amazon EC2 instance has maxed out on CPU. You try to visit your website and get an error. Now you've lost revenue and reputation.



© Logicworks. All rights reserved.

How can you prevent this from happening in the future?

Option 1: Resize your instance

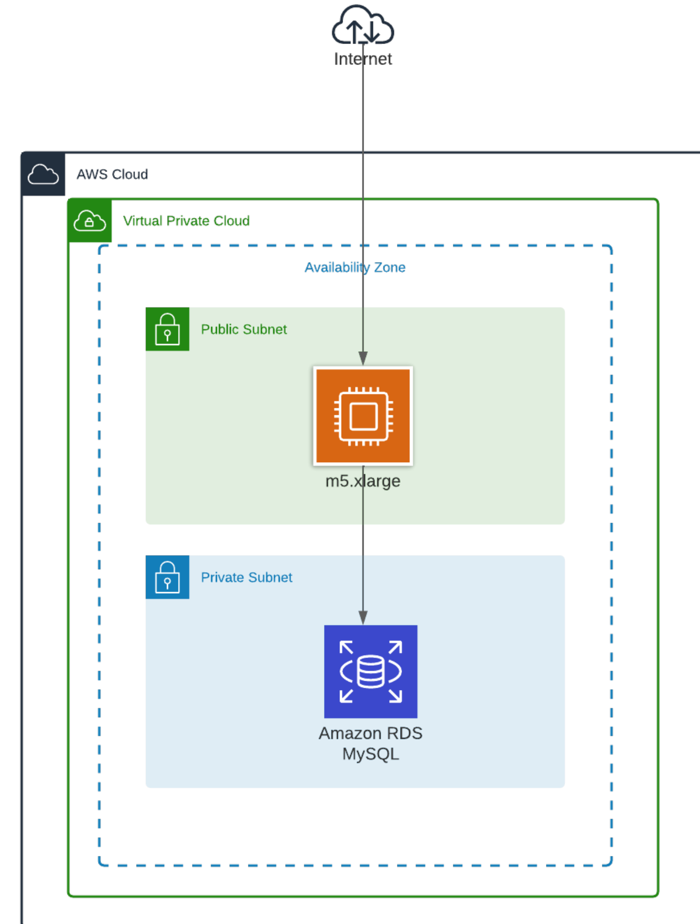
The traditional response to this failure would be to provision a larger server in your datacenter or colo facility. Similarly, you could decide that you're going to upgrade your m5.large to an m5.xlarge, which doubles your vCPU capacity. You take a snapshot, build an image, and relaunch as an m5.xlarge.

Pros:

- ✅ Easy to implement

Cons:

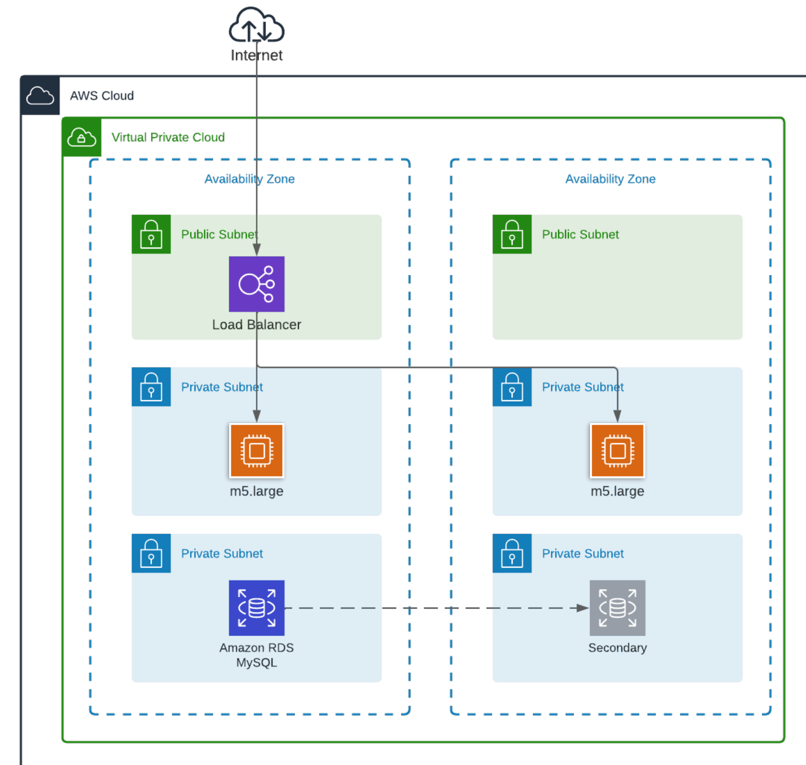
- ❌ Expensive: You're paying twice the cost of an m5.large, 24/7, that's $70/mo
- ❌ Might still not cover peaks: Just because you've increased the size of your instance, doesn't mean it accounts for all spikes in traffic. It reduces the likelihood of this happening, but is a "lazy" solution
- ❌ Single point of failure: If your instance fails, your entire application goes down



© Logicworks. All rights reserved.

Option 2: Run two instances behind a load balancer

In this second option, you can reduce the load on one single m5.large by running two m5.larges behind a load balancer (active-active). You can additionally improve resiliency at the same time by running those two instances across two different Availability Zones, so that there's redundancy on the datacenter level. You can configure your load balancer to direct equal amounts of traffic to both instances, and perform health checks so that if one fails, traffic is automatically redirected.



Pros:

- Multi-AZ setup + load balancer improves resiliency
  Load balancer in public subnet, instances in private subnet - improves security

Cons:

- Expensive: You're still paying twice the cost of a single m5.large
- Might still not cover peaks
- No automatic replacement of instance if one fails: If an instance fails, all traffic is directed to the other instance, but the failed instance must be rebooted manually

© Logicworks. All rights reserved.

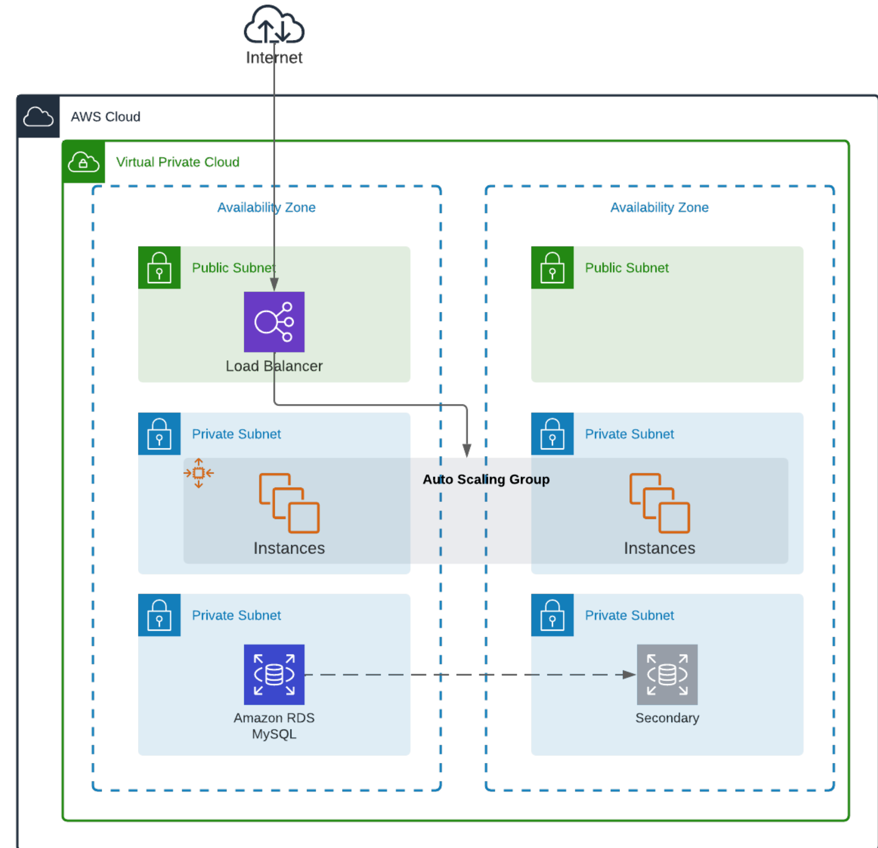
logicworks

Option 3: Auto Scaling

Neither resizing the instance nor running multi-AZ solves the problem of our spikey simple eCommerce application. Both solutions reduce the risk that a traffic spike will bring down your whole application, but they're both twice our original cost.

Enter Auto Scaling!

Rather than changing anything about our m5.large, we instead take an image of our existing instance and use that to create an Auto Scaling Group with a minimum of 1 and a maximum of 2 (or any other max that you feel is appropriate) and the condition that a scale-out should trigger when CPU utilization reaches 80% for a period of 5 minutes. Scale-in should trigger when CPU utilization reaches 40% for a period of 15 minutes.



© Logicworks. All rights reserved.

Let's say we've set up an Auto Scaling Group with a Simple Scaling policy (see next page for other Scaling types). You set it up with a minimum of 3 instances and a maximum of 6 instances, and other settings as seen in the screenshot below:

Configure group size and scaling policies Info

Set the desired, minimum, and maximum capacity of your Auto Scaling group. You can optionally add a scaling policy to dynamically scale the number of instances in the group.

Group size - *optional* Info

Specify the size of the Auto Scaling group by changing the desired capacity. You can also specify minimum and maximum capacity limits. Your desired capacity must be within the limit range.

Desired capacity
3

Minimum capacity
3

Maximum capacity
6

Health checks - *optional*

Health check type Info
EC2 Auto Scaling automatically replaces instances that fail health checks. If you enabled load balancing, you can enable ELB health checks in addition to the EC2 health checks that are always enabled.
EC2 ELB

Health check grace period
The amount of time until EC2 Auto Scaling performs the first health check on new instances after they are put into service.
300 seconds

When you launch the Auto Scaling Group, how many instances will it launch?
3 instances, since that's the minimum desired instance count.

CPU utilization is sustained over 80% for 3 minutes. What happens?
A scale-out event is triggered. The Auto Scaling Group will add 1 instance for a total of 4 instances.

CPU utilization is still sustained at over 80% for another 2 minutes. What happens?
Nothing. After a scale-out event, we've got a cooldown of 5 minutes. No more scaling event can happen during this cooldown period.

CPU utilization drops down to 30% for 10 minutes. What happens?
A scale-in event occurs. Instance count drops by 1, for a total of 3 instances.

CPU utilization remains at 30% for 10 minutes. What happens?
Nothing. 3 is the minimum instance count.

© Logicworks. All rights reserved.

Part 2: The Auto Scaling Process

1. Define your Launch Template

Every time you trigger a scale-out event, you need to tell your Auto Scaling Group what instance to create. You define this in a Launch Template. A Launch Template allows you to define:

- Amazon Machine Image ID
- Instance type
- Amazon EBS volume type and size
- Tags
- Keys to use for accessing the instance
- Networking - VPC, subnet, and security group

We'll go into detail about options for configuring Launch Templates in great detail on page 12.

2. Choose your Scaling Policy

a. **Target tracking scaling, a.k.a dynamic scaling:** Change capacity based on targeting a certain metric, like 70% CPU utilization. It works like a thermostat to automatically regulate capacity.

b. **Scheduled scaling:** Spin up and down resources based on a set schedule. For example, if you know your users are online at 9 A.M. every morning and log off at 5 P.M., then you can scale the Auto Scaling Group for your development resources beginning at 8:45 A.M. and begin to reduce capacity at 5:15 P.M..

c. **Predictive scaling:** Uses previous data to determine future scaling patterns (machine learning).

AWS has attempted to "wizardize" this process to make it easy to set up, and to some degree has limited configuration options.

© Logicworks. All rights reserved.

Auto Scaling groups (1)

Specify a scaling strategy for 1 Auto Scaling group.

☑ Include in scaling plan

Scaling strategy

The strategy defines the scaling metric and target value used to scale your resources.

- ○ **Optimize for availability** — Keep the average CPU utilization of your Auto Scaling groups at 40% to provide high availability and ensure capacity to absorb spikes in demand.
- ○ **Balance availability and cost** — Keep the average CPU utilization of your Auto Scaling groups at 50% to provide optimal availability and reduce costs.
- ◉ **Optimize for cost** — Keep the average CPU utilization of your Auto Scaling groups at 70% to ensure lower costs.
- ○ **Custom** — Choose your own scaling metric, target value, and other settings.

☐ Enable predictive scaling

Support your scaling strategy by continually forecasting load and proactively scheduling capacity ahead of when you need it. Info

☑ Enable dynamic scaling

Support your scaling strategy by creating target tracking scaling policies to monitor your scaling metric and increase or decrease capacity as you need it. Info

▼ **Configuration details**

ⓘ Below is the default configuration for the "Optimize for cost" strategy.

Scaling metric

Monitored metric that determines if resource utilization is too low or high. Info

Average CPU utilization

Target value

70 %

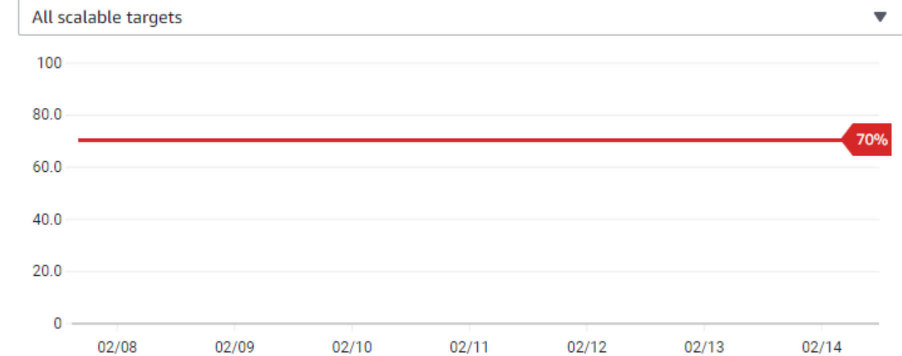
Replace external scaling policies

If your resources have existing policies, delete them and replace them with new target tracking scaling policies.

No

Average CPU Utilization (Percent)

All scalable targets ▼



© Logicworks. All rights reserved.

3. Set your thresholds and other configurations

Depending on what Scaling Policy you've chosen, you can now set which metric and threshold to trigger scale-in and scale-out events. There is no "recommended" threshold and policy, as this is highly dependent on a number of factors. This is where the "art" of Auto Scaling comes into play, and part of the reason why Auto Scaling isn't as simple as it seems.

Key Factors in Designing Auto Scaling Thresholds:

- **Operating system:** For example, if you're running a Windows Auto Scaling Group, and each instance takes 10 minutes to boot, you may want to set a lower threshold for scale-out (like 70%) in order to give you time for the instance to come up before traffic spikes further.
- **How long your instances take to boot:** If you're running sophisticated configuration management systems on boot, then be aware of how that affects boot times and change your thresholds accordingly.
- **Traffic patterns for your application:** If you have predictable traffic patterns with gradual traffic surges, then a Simple Scaling policy with 80% CPU utilization thresholds may work well. If your application is very spiky and spike durations are brief, then be aware that you may have to overprovision in order to have enough capacity when you need it, since any Auto Scaling action takes 2-5 minutes.

4. Set up Detailed Amazon CloudWatch monitoring

Next, you turn on Detailed Amazon CloudWatch monitoring. This means that rather than performing a health check on your target group every 5 minutes, you now perform a health check every 1 minute. This is highly recommended if you do any kind of Auto Scaling. It's a checkbox on the Auto Scaling Group configuration page.

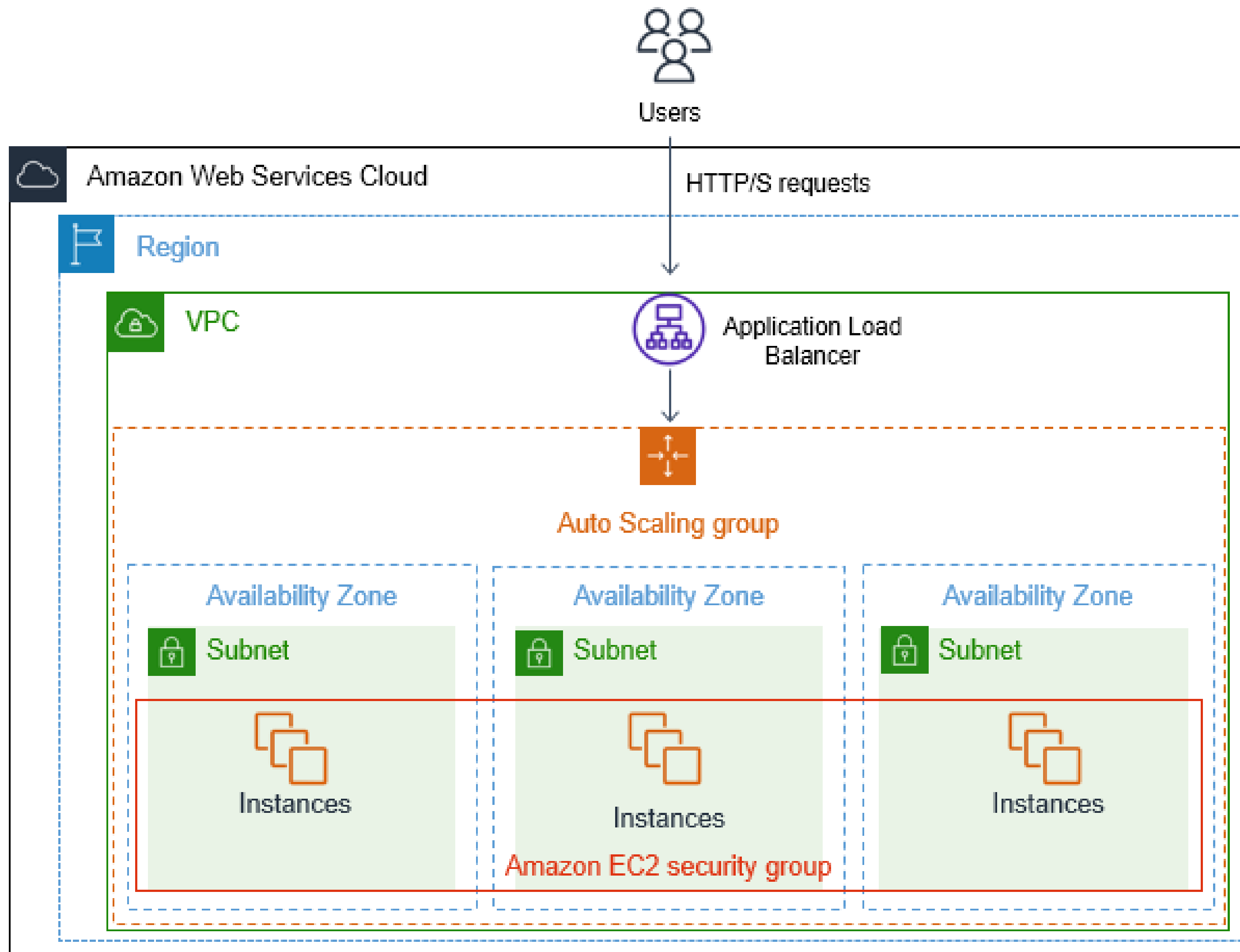
5. Test extensively

Set a conservative scaling policy to start, then track patterns for 2-3 months while refining scaling policies. Again, this work takes time, and may need refinement.

Try it Out: Auto Scaling Template

A tutorial with a set up very similar to the scenario described above can be found on next page – just review the documentation, and you'll have a simple Auto Scaling environment setup. Try to stop one of the two instances and see what happens.

© Logicworks. All rights reserved.



Source: AWS Tutorial

© Logicworks. All rights reserved.

Part 3: The Complexities of Auto Scaling

If you peruse the AWS Auto Scaling website, you'll see that AWS uses the terms "simple" and "easy" five times in the first two paragraphs. But any cloud administrator knows that a self-healing, fully scalable system requires significant engineering investment and experience.

Instance Bootstrapping

When a scale-out event is triggered, you need to supply AWS with an Amazon Machine Image (AMI) to build the instance. Maintaining the templates and scripts involved in the auto scaling process is no mean feat. This AMI can be built in dozens of different ways, but these methods can usually be categorized as follows:

Customize Marketplace base AMI with user-data

Use an AMI from the AWS marketplace, which usually has minimal modifications to the base operating system. Many of the companies we work with either use the Amazon Linux 2 AMI, a Windows Server AMI, or the Amazon ECS or EKS-optimized AMI as their base image. Then at launch, they pass through custom user-data to configure additional requirements, such as connecting to a configuration management tool, adding security agents (i.e. Antivirus and IDS), or adding additional packages.

This is by far the simplest option for bootstrapping, and works best in cases where customers are managing a relatively small number of instances (<100). This is because if you're adding user-data to an instance, you don't have a good way of tracking whether those configurations are working effectively or not. For example, it's very easy, for an antivirus agent install to fail and you'd never know!

Pros:

- ✔ Simplest option
- ✔ Fast boot times

Cons:

- ✖ Instance-by-instance configuration means lack of simple centralized control & management

© Logicworks. All rights reserved.

Use a Configuration Management Tool

Configuration management tool scripts can replace the need for a perfectly baked AMI. Instead, you can create a vanilla template with the minimum possible configurations that replaces your individual Golden AMIs for each server role. In this scenario, your instance userdata or boot script needs only to do what is necessary to connect to the configuration management master node.

Tools like Puppet, Chef (or AWS OpsWorks), or Ansible define everything on the servers from a single location, so there is a single source of truth about the state of the entire infrastructure. CloudFormation builds the foundation and installs the Puppet master, for example, and then Puppet attaches to the resources the node requires to operate such as Elastic IPs, network interfaces, or additional block storage. The final step is integration between the deploy process and auto scaling, where Puppet scripts automatically update Amazon EC2 instances newly added to auto scaling groups (due to instance failure, or scale events).

One challenge is that it may take upwards of several minutes for Puppet to run configurations and for the instance to take on its assigned roles. This can be an issue if you are experiencing load and need to quickly scale up your infrastructure. The solution for this is to create snapshot AMIs of already configured instances, and update your auto scaling groups to use those AMIs on startup.

Additionally, this process can be automated by using the AutoAMI Puppet module. This module will watch your Amazon EBS-backed instances to detect changes made by the Puppet agent, and automatically create AMIs that can be used to scale up or replace new instances during autoscale events.

The advantage of using a configuration management tool and building off of a base or marketplace AMI is obvious: if you are running 100+ machines, you update packages in a single place and have a record of every configuration change.

Build Your Own Image Dynamically

Why Immutable Infrastructure?

In reality, how you configure an instance depends on how fast the instance needs to spin up, how often auto scaling events happen, the average life of an instance, etc. In an Auto Scaling event, you often don't want to have to wait for your configuration

© Logicworks. All rights reserved.

management tool or any script to download and install 500MB of packages. In addition, the more tasks that the default installation process must complete, the higher the chance that something will go wrong.

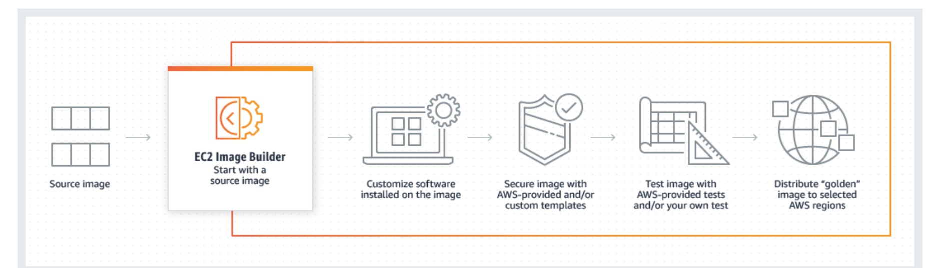
For instance, say that you update OpenSSL to the latest version every time your configuration management tool runs. Even though this happens very rarely, any number of random network issues could cause temporary outages while connecting to the package repository. If the initialization process does not fail elegantly, it could cost a lot of money: the instances keep dying and being recreated, in an hour it might spin up 30 instances and run up a substantial bill, especially if it is running large, production instances.

That's why companies who require the fastest possible Auto Scaling EC2 instance launch speeds select an "immutable infrastructure" methodology where they pre-bake AMIs. EC2 Auto Scaling speed is optimized because no additional configurations or application installations are needed at launch time. There are many other reasons why an immutable infrastructure approach is a great idea – stay tuned for a future eBook on this topic.

Image Baking Tools

The keys to this "immutable infrastructure" process are tools like Packer or AWS Image Builder (or, to plug our tool, Logicworks Image Factory!). Packer is a very popular option for companies who are already using other Hashicorp products, and AWS Image Builder is a great option if you want a simpler service that already has hooks into other AWS services.

These dynamic AMI building tools are designed to simplify the process of building, testing, and versioning AMIs. They usually provide a GUI for image building, and have hooks into other popular CI/CD and IaC tools, and are usually one element of a complex instance creation and code delivery process. Using this system, the launch template defines a pre-baked AMI



EC2 Image Builder. Source: AWS Documentation

© Logicworks. All rights reserved.

per compute role. For example, the “frontend webserver” role-specific AMI already will have nginx and the application’s frontend components installed and will have all O/S configurations (e.g. opened ports 80 & 443) complete.

Remember: It can take months for an experienced systems engineer to get comfortable working with image building, perfecting the scale conditions, and integrating it with existing instance launch pipelines. This is time that small engineering teams usually do not have, which is why many teams never reach the point of true auto scaling and instead rely on some combination of elastic load balancing and manual configuration. Allocating internal or external resources to create template-driven environments can decrease your buildout time by several orders of magnitude. This is why many IT firms devote an entire team of engineers to maintaining automation scripts, or leverage a partner like Logicworks.

Windows Auto Scaling

In addition to instance bootstrapping, another common complexity of Auto Scaling is Windows operating systems.

Windows users know that it can take 5-10 minutes for the Windows operating system to boot (or longer depending on application requirements!). This is an obvious issue for Auto Scaling Windows instances; depending on how frequently you’re performing health checks, it could be 2-5 minutes before a scale-out event is triggered, and then an additional 10 minutes before the Windows instance is ready to receive traffic. Depending on your application, this could be a minor inconvenience or a serious issue.

One way to get around this long boot time is to dynamically build an AMI that’s already sys-prepped so that when a scale-out event occurs, the instance comes online quickly. At Logicworks, we have a large enterprise customer who is currently using this method to Auto Scale a fleet of hundreds of EC2 instances. The company’s developers deploy application updates with TeamCity to QA, then to Production to a “reference” instance. After testing, the company creates an AMI from this machine and updates the corresponding Auto Scaling Group with the new AMI ID. The company leverages AWS SDK for .NET to create the AMI and update the corresponding Auto Scaling Group. This is a good solution for companies that have a large number of Auto Scaling Groups and frequent code updates.

© Logicworks. All rights reserved.

AWS also launched a service called Warm Pools, that allows you to launch and prep instances and then put them in a Stopped state until they're needed. The disadvantage of this system is that it's relatively complex to setup and troubleshoot, and you're still paying for attached storage on stopped instances.

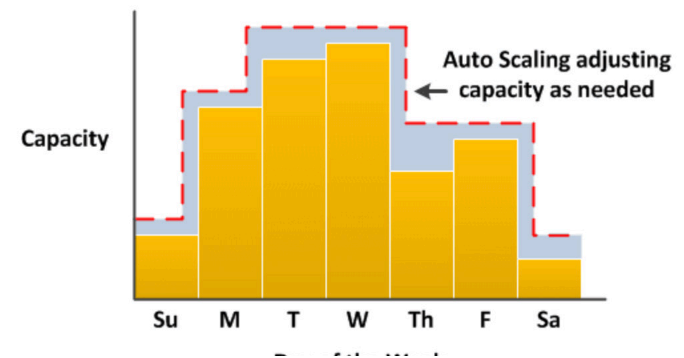
Spiky Applications

A common misconception about load-based auto scaling is that it is appropriate in every environment. In fact, some cloud deployments will be more resilient without auto scaling or on a limited basis. This is especially true for companies that have less than 50 instances, where closely matching capacity and demand has some unexpected consequences.

Let's say a startup has a traffic peak at 5:00 P.M. That traffic peak requires twelve Amazon EC2 instances, but for the most part they can get by with just two EC2 instances. They decide that in order to save costs and take advantage of their cloud's auto scaling feature, they will put their instances in an auto scaling group with a maximum size of fifteen and a minimum size of two.

However, one day they get a huge peak of traffic around 10 A.M., that is as high as their 5:00 P.M. traffic — but it only lasts for 3 minutes. Whether that traffic is legitimate or not, their website and application goes down.

Why does their website go down if they have auto scaling? There are a number of factors. First, their auto scaling group can only add instances every five minutes by default, and it can take 3-5 minutes for a new instance to be in service. Obviously, their extra capacity will be too late to meet their 10 A.M. spike. Because they do not have enough instances to handle the load, it not only triggers the creation of (non-helpful) extra instances, but the existing two servers are so overloaded that the health checks running on those instances start to slow down. When the Elastic Load Balancers see that the health check is not working, it drops the instance. This makes the problem worse and further increases load.



© Logicworks. All rights reserved.

In an ideal world, demand increases slowly and predictably. But sometimes big jumps happen over minutes, not days, and Auto Scaling doesn't always keep up. If this happens regularly, it would be wise to reexamine whether scaling down to two instances is a good idea. Even though the startup is saving money by scaling down to match capacity and demand, they are always at the risk of downtime.

In fact, it is generally true that auto scaling is most useful to those who are scaling to hundreds of servers rather than tens of servers. If you let your capacity fall below a certain amount, you are always going to be susceptible to downtime. No matter how the auto scaling group is set up, it still takes at least 5 minutes for an instance to come up; in 5 minutes, you can generate a lot of traffic, and in 10 minutes you can saturate a website. This is why a 90% scale down is almost always too much. In the above example, the startup should instead try to scale the top 20% of their capacity.

Summary

When AWS launched Auto Scaling in 2006, the industry proclaimed that the days of overprovisioning servers and capacity planning were over. After all, Auto Scaling embodies the full potential of the public cloud: to "pay for what you need and use, and not a byte more".

But as any experienced AWS user knows, Amazon EC2 Auto Scaling is complex. To create an automated, self-healing architecture that replaces failed instances and scales out with little or no human intervention requires a significant time investment upfront.

At Logicworks, we've spent the last 10 years perfecting our Auto Scaling principles – and every project is different, so we're continually learning, adapting, and innovating. If you have questions about your Auto Scaling practices, or want to conduct a Well-Architected Review with Logicworks, contact us.

© Logicworks. All rights reserved.

About Logicworks

Logicworks is a leading provider of AWS migration and managed services. As an AWS Premier Consulting Partner, we have helped hundreds of companies architect, migrate, and manage custom AWS environments. We specialize in complex, highly regulated workloads for healthcare, finance, and retail and have earned HIPAA, HITRUST, PCI-DSS, SOC1, SOC2, and ISO 27001 certification.

Learn more about Logicworks at www.logicworks.com.

"Given our business model of providing real-time data and analytics, the reliability of our platform is paramount. I cannot stress enough the role Logicworks plays in our effort to service our customers. They're more than a strategic partner, they're mission-critical."

Govi Rau, CEO
Noveda

© Logicworks. All rights reserved.