AWS Startups Blog

# Scaling on AWS (Part 3): >500K Users

by AWS Admin | on 29 DEC 2015 | in Startup Spotlight | Permalink | Share

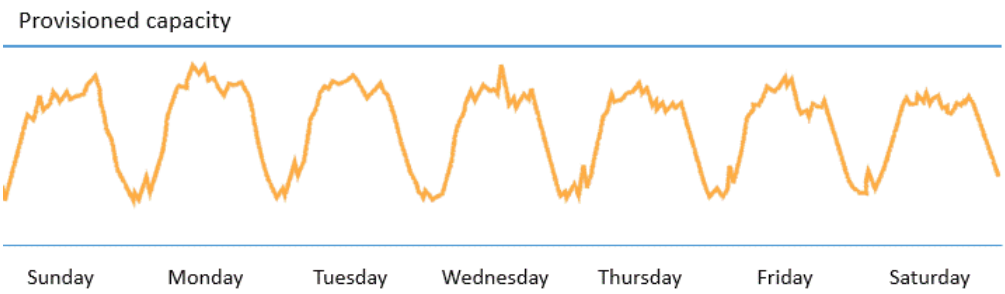
By David Kuo, Solutions Architect, AWS

This post is part 3 of a blog series that shows you how to iteratively evolve a basic AWS architecture into one that supports millions of users. In part 1 and part 2, I showed you how to begin scaling your architecture to handle more traffic, and I introduced AWS services that you can use to support over 10,000 or more users. In this post, I build on the same architecture to support 500,000 or more users.

The Auto Scaling concept I mentioned in the very first post will finally come into play, and I'll also show you how to use automation and monitoring services to manage a large environment.

**Auto Scaling**

After you shift workload components to the appropriate AWS services and decouple your application (as discussed in part 1 and part 2), you can introduce Auto Scaling to squeeze more efficiency out of your infrastructure. In the traditional infrastructure world where you have to provision a fixed number of servers, you are forced to stand up servers based on your peak demand. This method of server procurement forces you to do capacity planning and wastes money on unused server capacity. Let's take a look at the typical traffic experienced on an ecommerce website.

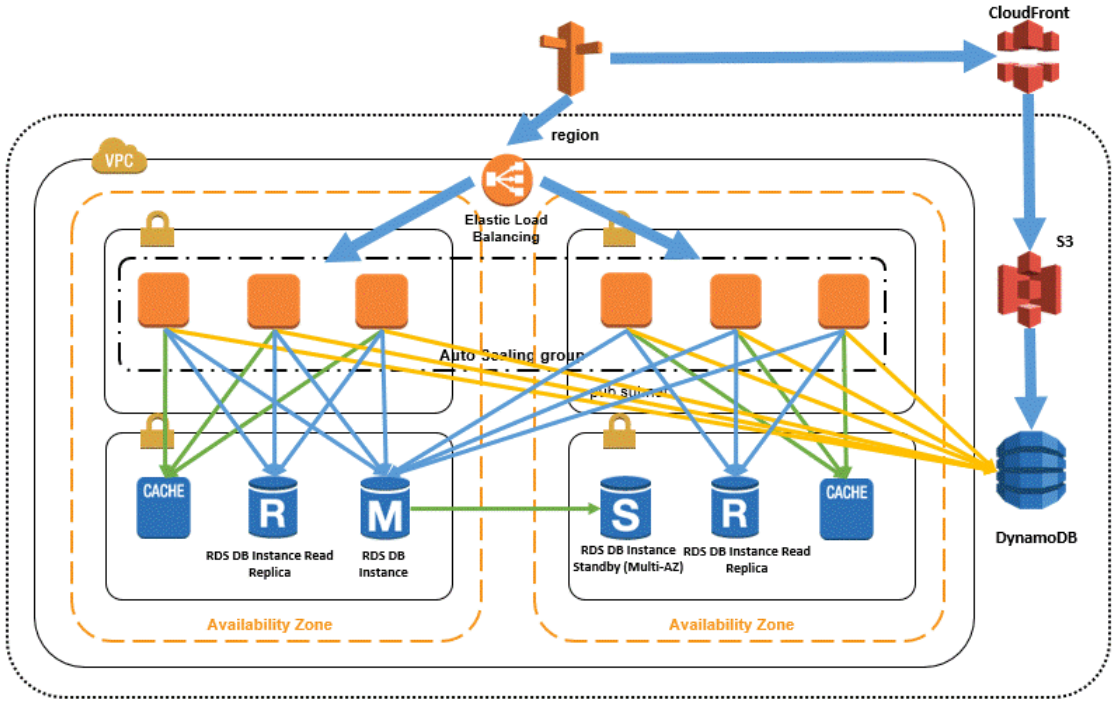



It is typical to see peaks and lulls throughout the week. As a business owner, how do you ensure a great user experience? To be safe, you would probably provision resources based on peak usage and eat the cost of idle servers during traffic lulls. You could hire engineers to monitor traffic patterns and create or decommission servers based on actual usage, but that is not efficient either. What happens when servers go down unexpectedly at 2:00 AM? How much negative business impact are you forced to absorb before an engineer becomes available to handle the incident? On the AWS platform, you can use Auto Scaling to automatically resize server fleets based on server metrics or a time schedule and replace unhealthy hosts. This feature not only removes some overhead of fleet management, but also empowers you to provision servers in a cost-effective way.

How does Auto Scaling know when to automatically resize a group of EC2 instances in a tier of your application? You can write policies based on metrics or a time schedule. For example, you could write a metrics-based policy that adds more EC2 instances when CPU utilization has been at or above 60% for the past five minutes. Alternatively, you could have a policy that provisions and ensures a fixed number of instances every weekday at 9:00 AM.

To get started, you create an Auto Scaling group and deploy EC2 instances within it. You can define a maximum and a minimum number of instances for an Auto Scaling group to meet your baseline performance requirements and enforce cost control. To create a template for the EC2 instances launched into an Auto Scaling group, you create a launch configuration to specify the instance type and Amazon Machine Image (AMI). You can configure Auto Scaling to use On-Demand instances or Spot instances. On-Demand Instances are charged based on hourly usage with no long-term commitment. Spot instances are instances you can bid on to take advantage of unused Amazon EC2 capacity and can cost much less than On-Demand instances at times.

The following illustration shows a simple, single-application tier that uses Auto Scaling to automatically scale web servers. You could have another Auto Scaling group for your application tier in private subnets as well. When used in combination with Elastic Load Balancing (ELB), Auto Scaling knows to register or deregister EC2 instances by using ELB-based health checks, and forwards traffic only to healthy EC2 instances.

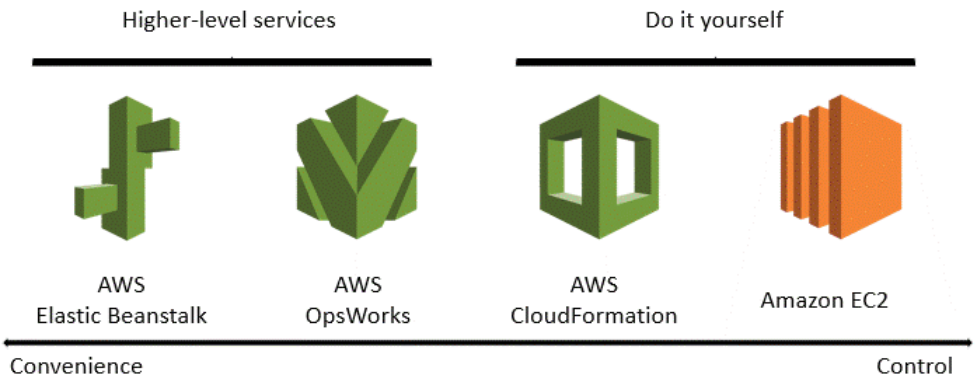



**Introduce Automation**

As your environment scales, it becomes less and less efficient to manually manage individual workload components. Not only does manual deployment takes time, but it also allows for human errors. You should automate as much of the application lifecycle as possible. Automation benefits include the following:

- Rapid changes
- Improved productivity
- Repeatable deployment
- Reproducible environments
- Leveraged elasticity
- Automated testing

Because customers have different needs when it comes to deployment automation, AWS provides a range of management services to let customers choose one that is most suitable. These services spread across a spectrum that varies on the level of convenience and control. AWS services that facilitate automation include AWS Elastic Beanstalk, AWS OpsWorks, AWS CloudFormation, and Amazon EC2.




AWS Elastic Beanstalk

AWS Elastic Beanstalk is a service that lets users easily deploy code written in Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, NGINX, Passenger, and IIS. Two primary concepts to understand when using Elastic Beanstalk are environments and versions.
An environment can be thought of as a container. It represents infrastructure automatically provisioned to run your application. When you create environments in Elastic Beanstalk, Elastic Beanstalk will take care of adding load balancing and Auto Scaling.

A version represents a flavor of your application code. This code can be stored in either Amazon S3 or GitHub. To get started with Elastic Beanstalk, you give it a version of an application to run and the environment to run it in. If you just want to deploy a simple two-tier web application and don't want to deal with setting up networking resources, Elastic Beanstalk is a great tool for doing that.

AWS OpsWorks

AWS OpsWorks provides an event-driven approach to application management. Various changes in the environment trigger events that are handled via custom code written as Chef recipes. When a lifecycle event is triggered, you can run either built-in recipes or custom recipes. In addition, AWS OpsWorks auto-heals your application stack, provides scaling based on time or workload demand, and generates metrics to facilitate monitoring. AWS OpsWorks gives you more control and granularity than AWS Elastic Beanstalk to help you deploy more sophisticated applications.

### AWS CloudFormation

AWS CloudFormation lets you provision AWS resources using a template in JSON format. You have the option to choose from a collection of sample templates to get started on common tasks, or create your own to describe AWS resources and dependencies required to support your application. With CloudFormation, you can modify environments in a controlled and predictable way. Because CloudFormation templates are text files, you can store them in your version control system and treat your infrastructure as code. You can version the infrastructure, view differences, and automate deployments using your existing continuous integration or continuous deployment pipelines. Recently, AWS introduced AWS CloudFormation Designer to help you visualize CloudFormation templates and edit them using a drag-and-drop interface.

Out of all AWS services that automate infrastructure deployments, CloudFormation gives you the most control and granularity. It empowers users familiar with the AWS platform to take advantage of all the services AWS has to offer.

### AWS CodeDeploy

I also want to mention AWS CodeDeploy, a service that automates code deployment to Amazon EC2 instances and instances running on premises. AWS CodeDeploy complements AWS Elastic Beanstalk, AWS OpsWorks, and AWS CloudFormation in that it automates the process of code deployment to existing infrastructure. You can use tags to designate deployment groups (e.g., development/QA/staging/production environments) and perform rolling deployments to minimize downtime. Deployments can be launched, stopped, and monitored through the AWS Management Console, AWS Command Line Interface (AWS CLI), API, or SDK.

CodeDeploy supports the concept of deployment health. You can specify the minimum number of instances that need to remain healthy, and CodeDeploy will stop a deployment if there are too many failed instance updates.
CodeDeploy also keeps track of deployment change history so you can see what versions are currently installed and the success rate of past deployments. Deployment events can be tracked down to the instance level.

CodeDeploy is platform and language agnostic. You use a configuration file to map files to destination hosts. You can also further customize the deployment process by registering commands to a set of standard events such as “install dependencies” or “stop server.” Commands can be code, script, a custom program, or even a management tool.

### AWS CodePipeline

Another code deployment automation tool is AWS CodePipeline. AWS CodePipeline is a continuous delivery service that lets you model a deployment process. You can customize four different stages—source, build, test, deployment—of the deployment process. For example, in the source stage, you can pull code from S3 or GitHub. In the build stage, you can use Jenkins or Solano as build servers. In the test stage, you can use BlazeMeter, Ghost Inspector, or Jenkins again. In the deployment stage, you can use Elastic Beanstalk or CodeDeploy. You can use some or all four stages to build a customized and automated deployment process.

**Using Metrics and Monitoring Tools**

You can’t improve what you don’t measure. It’s critical to use monitoring tools to collect internal and external system data to ensure that your system is behaving within expectation and, if not, to course correct. Internal system data, such as instance-level metrics, helps you to adjust the instance type based on actual resource consumption. Aggregate-level metrics from load balancers give you clues about how well the architecture is supporting your application needs. External system data lets you understand and improve the customer experience.

Following are some good categories of metrics to track and some services/tools available to capture these metrics:

• Host-level metrics—Amazon CloudWatch is a monitoring service that provides many useful EC2 instance-level metrics such as CPU utilization rate, disk read operations, and ingress/egress network traffic volume. These metrics allow you to continuously fine tune your choice of instance type and size. CloudWatch also provides metrics for Amazon Elastic Block Storage (Amazon EBS) volumes so you can see not only the number of average IOPS, but also the average payload size of each operation. CloudWatch also gives you the options to publish custom metrics and take advantage of the same visualization and alarming functionality as with standard metrics.

• Aggregate-level metrics—Amazon CloudWatch provides useful metrics on ELB load balancers so you can get an aggregate resource consumption view of the EC2 instances connected to the load balancers. For example, you gain visibility into the number of healthy or unhealthy instances, number of requests, latency, and number of 400 or 500 error counts. As you scale up to serve larger traffic, you’ll likely have load balancers in front of your EC2 instances. Reading ELB metrics along with host level metrics will give you a more complete picture of the workload resource utilization.

• Log analysis—Performing log analysis is critical to get a deeper understanding of your workload behavior. You can audit access, monitor resource consumption, and learn traffic patterns. For example, CloudWatch Logs can track the number of errors that occur in your application logs and send notifications. CloudWatch can also track API activity logged by AWS CloudTrail and fire alarms that way. If you need a feature that CloudWatch does not provide, you could look into third-party software such as Splunk and Sumo Logic.

• External site performance—Your application ultimately serves end users. If your end users encounter errors or their pages take five minutes to load, your priority should be fixing those issues. Therefore, understanding system performance from an end user’s perspective is a necessary reality check. Using a third-party web performance management tool like Pingdom can help you determine uptime and load time, monitor transactions, manage incidents, and fire informative notifications.

In the next post, which will also be the final post, I will continue to iterate the architecture with the intention to support millions of users. I’ll discuss big picture architecture strategies based on service-oriented architecture (SOA) and different database designs suitable for large amounts of data.

**Summary**

1. Auto Scaling is not the first step for scaling up AWS infrastructure. But when your infrastructure is ready, Auto Scaling removes fleet management overhead and automatically provisions just enough resources based on actual workload demand.