O'REILLY®

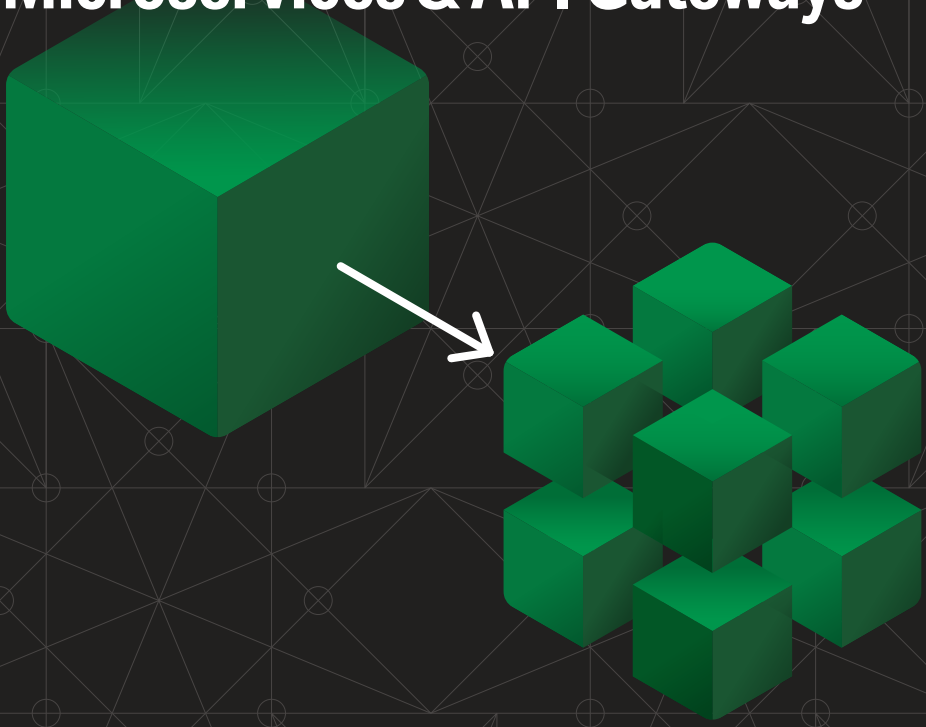
Compliments of NGINX

Load Balancing in the Cloud

Practical Solutions with NGINX and AWS

Derek DeJonghe

The NGINX Application Platform powers Load Balancers, Microservices & API Gateways



Load Balancing

Microservices

Cloud

Security

Web & Mobile Performance

API Gateway

FREE TRIAL

LEARN MORE

Learn more at nginx.com

NGINX

Load Balancing in the Cloud

Practical Solutions with NGINX and AWS

Derek DeJonghe

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Load Balancing in the Cloud
by Derek DeJonghe

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Virginia Wilson and Alicia Young
Production Editor: Nicholas Adams
Copyeditor: Jasmine Kwityn

Interior Designer: David Futato
Cover Designer: Randy Comer

May 2018: First Edition

Revision History for the First Edition
2018-05-08: First Release

This work is part of a collaboration between O'Reilly and NGINX. See our *statement of editorial independence*.

978-1-492-03797-2

[LSI]

Table of Contents

Preface

This book is for engineers and technical managers looking to take advantage of the cloud in a way that requires a load balancing solution. I am using AWS as the example because it is widely used, and therefore will be the most useful to the most people. You'll learn about load balancing in general, as well as AWS load balancers, AWS patterns, and the NGINX reverse proxy and load balancer. I've chosen to use NGINX as a software load balancer example because of its versatility and growing popularity. As adoption of NGINX grows, there are more people looking to learn about different ways they can apply the technology in their solutions. My goal is to help educate you on how you can craft a load balancing solution in the cloud that fits your needs without being prescriptive, but rather descriptive and informative.

I wrote this text to complement the AWS Quick Start guide to NGINX Plus. I truly believe in NGINX as a capable application delivery platform, and AWS as an industry leading cloud platform. That being said, there are other solutions to choose from, such as: Google Cloud, Microsoft Azure, Digital Ocean, IBM Cloud, their respective platform native load balancers, HAProxy, the Apache HTTP Server with the mod_proxy module, and IIS with the URL Rewrite module. As a cloud consultant, I understand that each cloud application has different load balancing needs. I hope that the information in this book helps you design a solid solution that fits your performance, security, and availability needs, while being economically reasonable.

As you read through keep your application architecture in mind. Compare and contrast the feature set you might need with the up-front and ongoing cost of building and managing the solution. Pay special attention to the automatic registration and deregistration of nodes with the load balancer. Even if you do not plan to auto scale today, it is wise to prepare with a load balancing solution that is capable of doing so to enable your future.

CHAPTER 1

Why Load Balancing Is Important

Load balancing is the act of distributing network traffic across a group of servers; a load balancer is a server that performs this action. Load balancing serves as a solution to hardware and software performance. Here you will learn about the problems load balancing solves and how load balancing has evolved.

Problems Load Balancers Solve

There are three important problem domains that load balancers were made to address: performance, availability, and economy.

As early computing and internet pioneers found, there are physical bounds to how much work a computer can do in a given amount of time. Luckily, these physical bounds increase at a seemingly exponential rate. However, the public's demand for quick complicated software is constantly pushing the bounds of machines, because we're piling hundreds to millions of users onto them. This is the performance problem.

Machine failure happens. You should avoid single points of failure whenever possible. This means that machines should have replicas. When you have replicas of servers, a machine failure is not a complete failure of your application. During a machine failure event, your customer should notice as little as possible. This is the availability problem: to avoid outages due to hardware failure, we need to run multiple machines, and be able to reroute traffic away from offline systems as fast as possible.

Now you could buy the latest and greatest machine every year to keep up with the growing demand of your user base, and you could buy a second one to protect yourself from assured failure, but this gets expensive. There are some cases where scaling vertically is the right choice, but for the vast majority of web application workloads it's not an economical procurement choice. The more relative

power a machine has for the time in which it's released, the more of a premium will be charged for its capacity.

These adversities spawned the need for distributing workloads over multiple machines. All of your users want what your services provide to be fast and reliable, and you want to provide them quality service with the highest return on investment. Load balancers help solve the performance, economy, and availability problems. Let's look at how.

Solutions Load Balancers Provide

When faced with mounting demand from users, and maxing out the performance of the machine hosting your service, you have two options: scale up or scale out. Scaling up (i.e., vertical scaling) has physical computational limits. Scaling out (i.e., horizontal scaling) allows you to distribute the computational load across as many systems as necessary to handle the workload. When scaling out, a load balancer can help distribute the workload among an array of servers, while also allowing capacity to be added or removed as necessary.

You've probably heard the saying "Don't put all your eggs in one basket." This applies to your application stack as well. Any application in production should have a disaster strategy for as many failure types as you can think of. The best way to ensure that a failure isn't a disaster is to have redundancy and an automatic recovery mechanism. Load balancing enables this type of strategy. Multiple machines are live at all times; if one fails it's just a fraction of your capacity.

In regards to cost, load balancing also offers economic solutions. Deploying a large server can be more expensive than using a pool of smaller ones. It's also cheaper and easier to add a small node to a pool than to upgrade and replace a large one. Most importantly, the protection against disasters strengthens your brand's reliability image, which is priceless.

The ability to disperse load between multiple machines solves important performance issues, which is why load balancers continue to evolve.

Evolution of Load Balancing

Load balancers have come a long way since their inception. One way to load balance is through the Domain Name System (DNS), which would be considered client side. Another would be to load balance on the server side, where traffic passes through a load balancing device that distributes load over a pool of servers. Both ways are valid, but DNS and client side load balancing is limited, and should be used with caution because DNS records are cached according to their time-to-live (TTL) attribute, and that will lead your client to non-operating nodes and produce a delay after changes. Server-side load balancing is powerful,

it can provide fine-grain control, and enable immediate change to the interaction between client and application. This book will mainly cover server-side load balancing.

Server-side load balancers have evolved from simply routing packets, to being fully application aware. These are the two types of load balancers known as network load balancers and application load balancers. Both named with respect to the layer of the OSI model to which they operate.

The application load balancers are where there are interesting advancements. Because the load balancer is able to understand the packet at the application level, it has more context to the way it balances and routes traffic. Load balancers have also advanced in the variety of features that they provide. Being in line with the presentation of the application, an application load balancer is a great place to add another layer of security, or cache requests to lower response times.

Even as load balancers have evolved, earlier "network layer" load balancers remain relevant even as newer "application layer" load balancers have also become useful. Network load balancers are great for simply and quickly distributing load. Application load balancers are important for routing specifics, such as session persistence and presentation. Later in this book, you will learn how all of these types of load balancing techniques work together to serve your goal of a highly performant, secure, and reliable application.

CHAPTER 2

Load Balancing in the Cloud

Cloud load balancing refers to distributing load across a number of application servers or containers running on cloud infrastructure. Cloud providers offer Infrastructure as a Service (IaaS), which renders virtual machines and network provisioning through use of an application programming interface (API). In the cloud it's easy and natural to scale horizontally as new application servers are just an API call away. With dynamic environments, where new machines are provisioned and decommissioned to meet user demand, there is a greater need for a load balancer to intelligently distribute traffic across your machines.

Load Balancing Offerings in the Cloud

In the cloud you're able to run any load balancer you'd like. Some load balancers, however, have a higher return on investment in regards to the solutions they provide versus the amount of time to integrate and manage. A great thing about the cloud is you can quickly proof of concept different architectures and solutions with little up-front investment. Let's take a look at the different types of load balancer offering in the cloud.

Native Cloud Load Balancers

Cloud-provided load balancers such as Amazon Elastic Load Balancer (ELB), Application Load Balancer (ALB), and Network Load Balancer (NLB) are built specifically for their environment. They require little up-front setup investment and little to no maintenance. Cloud providers such as Microsoft Azure, Google Cloud Compute, and Amazon Web Services each provide a native load balancer to use with their platform. These native load balancers integrate with the rest of their services, are inexpensive, and are easy to set up and scale. However, what the native cloud load balancers provide in ease of use and integration, they lack

in extensive features. These load balancers keep up with the dynamic nature of the cloud, but each of them only works within a single cloud provider.

Ported Solutions

What I'll refer to as ported solutions (not to be confused with portable solutions) are load balancing solutions that have been adapted from traditional hardware or virtual appliance offerings into the cloud. These appliance vendors have taken their load balancing offerings and made them into machine images available through cloud provider marketplaces. If the appliance you use is not already available, you can port solutions yourself by creating a cloud machine image from a typical disk image.

The great thing about ported solutions is that if you're already using the product in your data center, the configurations can be brought over with little effort. These ported solutions may be the same solution you're used to using in your current environment, and often have an extensive feature set. Many of these products are licensed and those licenses come with support models. It's important to note that the hardware acceleration which makes these solutions stand out as hardware appliances are not available when running virtualized or in the cloud.

There are a few pain points associated with ported solutions that make them less than ideal in a cloud environment: they can be difficult to automate, require extensive setup configuration, and do not scale well. These pains come from the fact that they're usually built on obscure or specialized operating systems that are optimized for networking, which is great, but can make using configuration management difficult or impossible. Many of these companies understand those difficulties and have introduced APIs through which you can configure their product. The APIs make using the product in the cloud much more palatable, but causes your infrastructure automation to need an outside actor making those API calls, which in turn adds complexity. Some organizations value reusing the same configuration, the comfort of sticking with a familiar product, vendor, and support team, over the ease of automation and integration. This sentiment is valid because change takes time and in turn has costs.

Software Load Balancers

Software load balancers can be installed on top of common operating system distributions. Software load balancers offer extensive feature sets, ease of configuration, and portability. Because these load balancers can be installed on the same operating system your team is using for your application stack, you can use the same configuration management tool you use to manage the rest of your environment. Software load balancers can exist anywhere, because it's just software you can install on bare-metal servers, virtual machines, containers, or even work-

stations, with as little or as much power as you need. You're able to push your configurations through a full continuous integration and continuous delivery (CI/CD), system just as you would your application code. This process is valuable because it's important to test integration between your application and a load balancer, as the load balancer has an influence on delivering an application. The closer to the development cycle these integration tests catch issues, the less time and money they take to correct. Software load balancing provides flexibility, portability, and feature set.

Each of these offerings has its perks. It's up to you to choose which are most important for your team and application needs. You also don't need to pick just one, it's common to see cloud-provided load balancers fronting an array of more sophisticated load balancers to take advantage of all features available.

Global Load Balancing with Route 53

In some cases, application stacks are not located in a single geographic location but rather are spread out into multiple installations across the globe. The global distribution I'm referring to is not for disaster recovery; it's for reduced latency and sometimes legal requirements.

The AWS Route 53 DNS service offers features to enable you to direct users to the closest installation of your service based on latency or to specific endpoints based on the geographic location of the request origin. Route 53 is able to perform this routing on DNS records that utilize the routing features named Geolocation and latency-based routing.

Geolocation routing allows you to specify endpoints for geographic locations by continent, by country, or by state in the United States. It works by mapping the IP of the entity requesting the resource to a location. Once the requester's location is found, it is mapped to the most specific geographic location endpoint that you've configured.

When using the latency-based routing feature, Route 53 detects the closest region based on DNS request latency. To utilize this feature you must have EC2 endpoints in multiple regions. Upon receiving the request, Route 53 determines which region gives the user the lowest latency, and directs the user to that region. It is important to note that a lower-latency region will win out over geographic proximity.

When using Route 53 global routing, it's important to consider where your users' data is stored and how user travel may impact their usage of your application. In other cases, where content is required to be geographically restricted, you may intend for users to not be able to access specific content outside of a given location.

Cloud Considerations for Load Balancing

Common cloud patterns, such as scaling server tiers, enable abilities like self-healing infrastructure and adjusting capacity to meet demand. However, these abilities bring about additional considerations for load balancing in the cloud. Let's take a look at some of these in the following sections.

Scaling Infrastructure

Scaling infrastructure is the most important consideration to have when load balancing in the cloud. To take full advantage of the cloud your application infrastructure should increase and decrease capacity automatically to match the demand of your users. Your load balancer must be able to register and deregister nodes from load balancing pools though automation. This task needs to be done through automation to support scaling without human interaction. Common ways of doing this are load balancer APIs, configuration templating and seamless reloads, service discovery integration, and DNS SRV records.

Scaling Load Balancers

It's not just your application that should be capable of scaling, but also your load balancer. Load balancers are often the front door of your application—they need to be highly available and always accepting connections. Rather than over provisioning only a couple load balancers, you can auto scale them to match the capacity you need when you need it, and release that capacity when you don't. The load balancer you choose for your cloud environment should be able to run as an array of servers.

Health Checks

While cloud virtual machines are reliable, planning for success is done by planning for failure. Your application stack will experience issues, employing your load balancer to health check the application and only pass traffic to healthy nodes ensures that your end users see as little interruption as possible. Health checks should be configurable for the request type, response, and timeout. With these settings you can ensure that your application is responding correctly, and in a reasonable amount of time.

Further Reading

- Amazon Elastic Load Balancing

CHAPTER 3

NGINX Load Balancing in the Cloud

It's hard to know how to apply this advice without a concrete example. I've chosen to focus on NGINX and AWS to show you precisely how you might go about load balancing in the cloud because of their existing and growing market share in the load balancing space. NGINX is more than just a load balancer, it's a full application delivery controller. Application delivery controllers are load balancers that have an advanced set of options that allow you to fully control how your application is delivered. NGINX is a software load balancer that brings a lot to the table. This chapter will introduce you to a subset of important features of NGINX, its portability, and its scaling capacity.

Feature Set

NGINX is free and open source software; NGINX Plus is a licensed option that offers advanced features and enterprise-level support. In this section I will outline features of NGINX, and NGINX Plus, and I will specify when examples are paid features of NGINX Plus.

The features that are included in NGINX, the open source solution, are fully capable of delivering your application. NGINX Inc. has stated that they will continue to maintain and develop the open source code base. At its core NGINX is a reverse proxy and load balancer. NGINX can load balance both TCP and UDP, and has specific modules for HTTP(S) and Email protocols. The proxy features provide a basis for NGINX to control requests, responses, and routing for an application. There are numerous load balancing algorithms available in NGINX, as well as passive health checks. NGINX is able to perform layer 7 routing, which is necessary for microservice architectures that provide a uniform API endpoint. NGINX provides a set of connection settings for optimizing client- and server-side connections, which include keepalives, SSL/TLS listener and upstream settings, connection limits, and also HTTP/2 support. NGINX can also handle

caching static and dynamic content. All of these features contribute to tuning your application delivery and performance.

Extensibility

NGINX is extremely extensible. You can extend NGINX with modules written in C. The community and NGINX Inc., have many modules to choose from, if you don't see what you're looking for you can create your own, or use an embedded scripting language. NGINX Inc., introduced nginScript in 2015 as a module for NGINX and NGINX Plus. nginScript is a JavaScript implementation that extends the NGINX configuration providing the ability to run custom logic against any request. There are other runtimes available for NGINX, such as Perl, and the most popular, Lua. The Lua module is open source and was started in 2010 and developed by Yichun Zhang. These extensibility options make the possibilities of NGINX limitless.

Security Layers

NGINX, the open source solution, also provides a complete set of security features. NGINX comes with built-in support for basic HTTP authentication, and IP-based access control. For more advanced authentication, NGINX is able to make subrequests to authentication providers such as LDAP or custom auth services. Securing static content through the use of shared secret hashing allows you to provide limited time access to resources; this feature is known as secure links. NGINX can also limit the rate of requests from a client to protect against brute-force and denial-of-services attacks.

Effective security is done in layers; NGINX offers plenty of layers to provide your application with security. With NGINX you can take it even further by building a module that incorporates ModSecurity 3.0 into NGINX. ModSecurity is the go-to solution in open source web application security. Starting in version 3.0, ModSecurity runs natively with NGINX to provide security against layer 7 application attacks such as SQL injection, cross-site scripting, real-time IP blacklisting, and more. This module effectively turns your NGINX implementation into a Web Application Firewall (WAF). You can build ModSecurity yourself as a dynamic module for the open source NGINX, or use a prebuilt module available with NGINX Plus subscriptions and support.

Additional NGINX Plus Features

NGINX Plus comes with some features that are more specialized to work with your application. NGINX Plus provides advanced features for session persistence, a response time–based load balancing algorithm, active health checks, and advanced cache control. NGINX Plus is also able to use DNS SRV records as server pools, which enables a seamless integration with a service discovery

system. You can use NGINX Plus to facilitate live media streaming and MP4 streaming bandwidth control. The NGINX Plus solution offers an API for live, on-the-fly, reconfiguration of NGINX Plus servers and configuration sharing among NGINX Plus nodes, so you only have to make your API calls to the NGINX Plus Master. The NGINX Plus API also enables use of an integration with AWS Auto Scaling, to automatically register and deregister your auto scaling instances with your NGINX Plus load balancer.

NGINX Plus has advanced authentication features, such as the ability to authenticate users based on a JSON Web Token (JWT). With this feature you're able to verify or decrypt the token, once the token is verified, you can use the JWT's claims to more intelligently make access control decisions directly in your load balancer. This ability enables NGINX Plus to be used as an authenticated application gateway. An example of a commonly used JWT authentication is OpenID Connect, which means if you're already using OpenID, your authentication will seamlessly integrate with NGINX Plus.

With all these available features, NGINX and NGINX Plus really do live up to the name Application Delivery Controller. NGINX is fully capable of being tuned to increase your application's performance, enhancing its security, and being extended to run custom logic. All of these features make NGINX and NGINX Plus capable choices to be the ingress for your application. These features enable you to have full control over requests coming from the internet into your environment, how they're routed, who gets access, and how they access it.

Portability

NGINX and NGINX Plus can be run anywhere, because they're just software. This means you can run NGINX in your data center, in any cloud environment, on any distro, as well as in containers.

In the data center or in the cloud you can run NGINX on top of any Linux/Unix distribution as they all have access to a C compiler so you can build the open source package yourself. For many of the main line Linux distributions, such as Debian, Ubuntu, RHEL, Centos, and SUSE, there are prebuilt packages available from NGINX repositories. Many cloud providers, such as AWS and Azure, have marketplaces where a prebuilt machine image with NGINX already installed is available. There is an NGINX version for Windows, however, it is considered beta because of a number of limitations and other known issues.

NGINX is also able to be run inside of containers, like Docker, LXC the Linux container, and rkt (pronounced rocket). The most common of these container types being Docker. The appeal of containers is that they're self-contained and portable. NGINX and NGINX Plus can both be built into custom container images, or you can pull official NGINX container images from Docker Hub. The

official NGINX container is based on Debian, with the option to pull a version built on Alpine Linux for a more lightweight container.

The official Docker Hub repository also provides an option to pull an image with the Perl module built in, as it's common to use this module to inject environment variables into your NGINX configuration. This practice is extremely valuable because it enables you to use the same container image between environments ensuring that your configuration is thoroughly tested.

Portability is very important—the ability to run NGINX anywhere means that you're able to test and run environments everywhere. Whether you intend to run smaller environments closer to your users or disaster recovery environments on different hosting providers, software load balancers like NGINX are flexible enough to enable your goal.

Scaling

Cloud infrastructure is meant to scale with demand of your application; your load balancing solution must be able to keep up. Today, the internet is bigger, and users demand faster performance than yesterday. NGINX was architected from the initial design with performance in mind. The NGINX process fulfills the work for each connection in an asynchronous non-blocking fashion. A properly tuned NGINX server can handle hundreds of thousands of connections for a single process pinned to a single CPU core. In the event that your user base breaches those bounds NGINX also flawlessly scales horizontally.

The creator of NGINX, Igor Sysoev, found that most of the time spent in network applications was not in the processing, but in waiting for clients to fulfill their portion of the connection. With this information Igor concluded that he could dramatically increase performance by fulfilling the server side of the connection and moving on to other portions of work while the client fulfills their portion. This is the asynchronous architecture that enables NGINX to fully utilize the processing power of the machine to serve hundreds of thousands of requests in parallel. This power scales vertically by adding more CPU cores to the node and binding an NGINX worker process to each core available. It's recommended to not run more NGINX processes than cores because the NGINX process is capable of fully utilizing the core it's pinned to.

While a single NGINX server is fully capable of handling a massive amount of client connections on modern hardware, NGINX is also able to be scaled horizontally by balancing load over multiple NGINX machines. Being able to scale horizontally is important, first for high availability, second for cost savings. Scaling horizontally protects you from single server or data center outages. Cost savings comes into play when you scale out to meet demand and scale in when your demand is lower, which allows you to only pay for what you use. In Chapter 5

you will learn more about running multiple NGINX servers in parallel and balancing load between them.

The capability of a single NGINX process has proven that its architecture was designed for performance. An NGINX server can have many worker processes to scale capacity vertically within a node. Scaling NGINX horizontally by running multiple nodes is also possible. NGINX is just one way to approach this but it's a great example of how to do it well because of its feature set, scaling capability, and portability. In the next chapter, you will learn about load balancing considerations for an auto scaling application layer.

CHAPTER 4

Load Balancing for Auto Scaling

The purpose of auto scaling is to increase or decrease the number of virtual machines or containers as dictated by the scaling policy. A scaling policy can be triggered by many events, such as CloudWatch metric alarms, a schedule, or anything that is able to make an API call. This enables your application to scale its capacity based on real-time demand or planned usage. As capacity is increased or reduced, however, the nodes being added or removed must be registered or deregistered with a load balancing solution. For auto scaling to function properly all aspects of this process must be automated. In this chapter you will learn what to consider when load balancing over auto scaling applications, plus what approaches you may take to address these considerations.

Load Balancing Considerations for Auto Scaling Groups

As you've already learned, auto scaling implies that nodes are automatically created or removed. This action may be as a result of utilization metrics or schedules. Machines or containers being added will do nothing to serve more load, unless the entity that is feeding them load is in some way notified. When machines or containers are removed due to an auto scaling event, if those nodes are not deregistered, then the load balancer will continue to try to direct traffic to them. When adding and removing machines from a load balancer, it's also important to consider how the load is being distributed and if session persistence is being used.

Adding Nodes

When a node is added to the application's capacity it needs to register with the load balancing solution or no traffic will be pushed to this new capacity. It may seem that adding capacity and not using it may be harmless, but there are many adverse effects it may have. Cost is the first, don't pay for things you're not using.

The second has to do with your metrics. It is common to use a statistical average of CPU utilization to determine if capacity needs to be added or removed. When using an average, it is assumed that the load is balanced among the machines. When load is not properly balanced this assumption can cause issues. The statistic may bounce from a high average (adding capacity) to a low average (removing capacity). This is known as rubber banding; it takes action to serve demand but does not actually provide the intended effect.

Deregistering Nodes

When auto scaling, you must also consider nodes deregistering from the load balancer. You need nodes to deregister from your load balancer whether they're actively health checking or not. If a node suddenly becomes unavailable your clients will experience timeouts, or worse session loss if your application depends on session persistence. To cleanly remove a node from an application pool it must first drain connections and persistent sessions then deregister.

Algorithms

The load balancing algorithm being used by your solution should also be considered. It's important to understand how adding or removing a node from the pool will redistribute load. Algorithms such as round robin aim to distribute load evenly based on a given metric; round robin balances based on the request sum metric. Adding and removing nodes when using an algorithm like this will have little impact on the distribution. In algorithms that distribute load by using a hash table, it's possible that similar requests will be direct to the same server. In a static environment this type of algorithm is sometimes used to provide session persistence, however, this can not be relied on in an auto scaling environment. When adding or removing nodes from a load balancer using a hashing algorithm, the load may redistribute, and requests will be directed to a different server than before the capacity change.

When load balancing over Auto Scaling Groups you have a number of things to consider. The most important of these being how machines are registered and deregistered from the load balancing solution. A close second being the impact on session persistence, and how your load distribution will change.

Approaches to Load Balancing Auto Scaling Groups

The considerations outlined in the previous section can be approached with a bit of automation and foresight. Most of the considerations that are specific to the auto scaling pattern have to do with automatic registration and deregistration with the load balancer.

Proactive Approach

The best way to approach the session persistence issue is to move your session. To load balance appropriately, your client should be able to hit any node in your application tier without issue. You should store session state in an in-memory database, such as Redis or Memcached. Giving all application servers access to centralized, shared memory, you no longer need session persistence. If moving the session is not possible, a good software or ported load balancer will allow you to properly handle sessions.

An automatic registration and deregistration process is the best approach for load balancing over auto scaling tiers. When using Auto Scaling Groups, or Elastic Container Service (ECS) Service Tasks, there is an attribute of those resources that takes a list of ELBs, or Target Groups for use in ALB/NLBs. When you provide an ELB or Target Group, the Auto Scaling Group, or Container Service will automatically register and deregister with the load balancer. AWS native load balancers can drain connections but do not drain sessions.

When using something other than a cloud-provided load balancer you will need to create some sort of notification hook to notify the load balancer. In AWS there are three ways to do this: the node transitioning states makes a call to the load balancer; the load balancer queries the AWS API regularly; or a third-party integration is involved. The load balancer being used must be able to register or deregister nodes through automation. Many load balancing solutions will offer an API of some sort to enable this approach. If an API is not available, a seamless reload and templated configurations will work as well.

The best way for an Instance in an Auto Scaling Group to register or deregister from a load balancer as it comes up or prepares to go down is through Lifecycle Hooks. The Lifecycle Hook is a feature of AWS Auto Scaling Groups. This feature creates a hook between the Auto Scaling Groups' processes and the OS layer of the application server by allowing the server to run arbitrary code on the instance for different transitioning states. On launch the Auto Scaling Group can signal a script to be run that will make a call to the load balancer to register it. Before the Auto Scaling Group terminates the instance, the lifecycle hook should run a script that instructs the load balancer to stop passing it new traffic, and optionally wait for connections and sessions to drain before being terminated. This is a proactive approach that enables you to ensure all of your client connections and sessions are drained before the node is terminated.

Reactive Approaches

You can also use a reactive approach by having your load balancer query the AWS API and update the load balancer as nodes come online or are removed. This approach is reactive because the load balancer is updated asynchronously

after the node is alive or already gone. A reactive approach falls short because the load balancer will inevitably experience timeouts before health checks deem the missing node unhealthy, and does not take session persistence into account. There are load balancing features that will gracefully handle upstream timeouts and proxy the request to the next available upstream node; this is necessary to prevent your clients from seeing HTTP errors. Reactive approaches can also be done via Simple Notification Service (SNS), notifications triggered by the Auto Scaling Group or the ECS Service. These notification are able to trigger arbitrary code to be run by AWS Lambda Functions or an API that controls registration and deregistration.

At this time the AWS native load balancers only support round-robin load balancing algorithms. If you're using a load balancer that supports other algorithms based on the pool, such as a hashing algorithm, you should consider if rebalancing is of concern. If redistributing the hash and rebalancing will cause problems with your application, your load balancer will need a feature that will intelligently minimize the redistribution. If these features are not available you will need to evaluate if session persistence and connection draining will better fit your needs.

Now that you understand the considerations of auto scaling, and some insight on how to approach these considerations, you can build your own auto scaling tier in AWS.

Further Reading

- Amazon Auto Scaling
- Amazon Elastic Container Service
- Amazon Simple Notification Service
- Amazon Lambda
- Auto Scaling Lifecycle Hooks

CHAPTER 5

NGINX Plus Quick Start and the NLB

Amazon Web Services has a number of quick start templates to help you bootstrap a demo environment in just a few clicks. AWS has created a quick start that creates a demo environment with NGINX Plus load balancing over two separate applications. The quick start demonstrates routing to different applications based on URL path, round-robin load balancing, and auto registration for Auto Scaling Groups. This chapter will briefly describe what a quick start is, and what the example NGINX Plus quick start builds.

AWS released their Network Load Balancer, NLB, offering in 2017. The NLB is a layer 4 load balancer, meaning this load balancer operates at the transport layer of the OSI model. The transport layer is responsible for reliable transmission of communication over the network; this is the layer in which TCP and UDP operate. This load balancer is unlike AWS's other load balancing offerings, and this chapter will also show you what makes the NLB different.

Quick Starts and CloudFormation

Amazon quick starts utilize the AWS CloudFormation service to produce an environment for demonstration. These quick starts are intended to exhibit good practice on AWS cloud with as little effort on the user's side as possible. Amazon wants to show the power and possibilities of their platform without you having to spend an entire day setting things up.

AWS CloudFormation is a service that translates declarative data objects into living AWS resources. These data objects are called templates because they take input parameters and use functions that can alter what they produce. CloudFormation templates can be used over and over to produce separate environments and can be as all inclusive or as modular as you like. I highly recommend using a declarative infrastructure management tool such as CloudFormation to manage

your cloud environment. The templates used by these tools can be stored in source control, enabling you to version, audit, and deploy your infrastructure like software.

The Amazon quick start templates are publicly available on S3 as well as on GitHub and they're all fantastically documented with a quick start guide. These guides will walk you through booting the quick start in just a few simple clicks by use of CloudFormation. The quick starts do tend to default to smaller resource sizes for demonstration because you are ultimately responsible for the cost incurred by booting quick start templates into your AWS Account.

The AWS NGINX Plus Quick Start

The quick start guide for setting up the NGINX Plus Quick Start can be found at *https://aws.amazon.com/quickstart/architecture/nginx-plus/*. This guide will thoroughly walk you through booting the quick start template.

After the guide walks you through preparing your account, you will instantiate the CloudFormation stack that creates the environment. The CloudFormation stack produces a Virtual Private Cloud, an ELB, and three Auto Scaling Groups. One Auto Scaling Group is a group of NGINX Plus instances, and the other two are NGINX web servers that mimic two separate applications. The quick start template builds other CloudFormation stacks, which modularizes their CloudFormation code for re-usability. This pattern is called nesting, and is much like using an include to import a library in software.

The Virtual Private Cloud (VPC) is a virtual network spanning two physically separated data centers known as Availability Zones, or AZ. The VPC is divided into public and private subnet pairs, with one of each in each AZ. This CloudFormation stack also includes all the necessary networking components such as proper access to the internet, routing, network access rules, and DNS.

After the network is created, the template builds another stack including a Linux bastion EC2 Instance, and security group, allowing you SSH access to the environment. The Linux bastion attaches an Elastic IP address within a bash bootstrap script. The bastion is a typical practice and provides you a secured access point into the network.

The last stack that is created by the quick start is the NGINX Plus demonstration. Within this stack are three Auto Scaling Groups. One Auto Scaling Group is an NGINX Plus load balancer, and the others are NGINX web servers posing as web apps. In front of the NGINX Plus Auto Scaling Group is an Internet-Facing Elastic Load Balancer. The NGINX Plus machine runs a small application named nginx-asg-sync that looks for Auto Scaling Group changes and notifies the NGINX Plus API for auto registration.

The guide will walk you through exercising the demonstration. You will see how NGINX Plus is able to load balance over the application, and route to different application servers based on the URL path. The demo also provides some example configurations that can enable the NGINX Plus status page and on-the-fly reconfiguration API.

NGINX and the NLB

The Amazon Web Services Network Load Balancer (NLB) is a layer 4 load balancer. In the quick start, an Elastic Load Balancer, a layer 7 load balancer, was used. Layer 7, the application layer, of the OSI model is the layer in which protocols such as HTTP, DNS, NTP, FTP, operate. There are some important differences between the two, and when load balancing over NGINX, the NLB is preferred. This section will explore some of the differences between the two, and the advantages of the NLB for load balancing over NGINX.

The Elastic Load Balancer, being a layer 7 load balancer, can do things like SSL/TLS termination, add cookies and headers to HTTP requests, and acts as a proxy. These features are redundant when working with NGINX. In fact, when any service is behind a layer 7 load balancer there are certain considerations to be had, because the services client is actually the load balancer directly in front of them, which then serves the end user. For these scenarios there is a standard defined as the PROXY Protocol that addresses the difficulties of being behind a proxy, however, you do have to configure your service to rely on the standards in this protocol.

The Network Load Balancer does not act as a proxy and therefore does not manipulate the request. This type of load balancer simply facilitates the connection across a pool of servers. When the NGINX node receives the connection it appears to be directly from the end client. With this topology you're able to configure NGINX the same way you would if it were at the edge, with all the advantages of active-active highly available NGINX load balancers. The NLB is also the first Amazon load balancer to allow for connections living longer than an hour, Elastic IP support, and UDP support.

Some of the advantages of the NLB, are that they're capable of being assigned Elastic IP addresses, the pool targets can be of dynamic port, and there's great health check support. Elastic IP support is important because in some industries it is still common to find static IP requirements, and therefore the other AWS load balancers cannot be used because they have dynamic public IP addresses. Targets being registered by IP and port is important because an IP can be registered multiple times with different ports, which is a big deal for container clusters. This enables clusters to run multiple containers of the same service on a single instance. Health checks are able to be done as TCP/UDP connection tests,

or as layer 7–aware HTTP requests and validation. Lastly it scales like an Amazon product, able to handle massive spikes and perform without delay.

With these features and advancements, the NLB makes for a great Amazon native load balancer in front of NGINX, and a great addition to the Amazon line of load balancers. There were a lot of features that were first seen in the NLB which enables more organizations to take advantage of these load balancers that integrate so well with the rest of the Amazon ecosystem.

Further Reading

- Amazon CloudFormation
- NGINX Deployment Guide: AWS High Availability Network Load Balancer

CHAPTER 6

Monitoring NLBs and NGINX Plus

Monitoring your application is almost as important as building it, because monitoring provides the measurement for successfully maintaining and adjusting systems and applications. As the systems world gets more and more automated, good monitoring is indispensable, as the metrics collected by monitoring serve as the basis for automated responses. In order to be able to automate responses to changes in your application stack, you must collect metrics. The more metrics you collect, the more informed your automated responses can be. This chapter demonstrates different ways to monitor the AWS Network Load Balancer, and our example software load balancer NGINX and NGINX Plus.

CloudWatch for Monitoring

AWS's native monitoring solution is called CloudWatch. CloudWatch ingests metrics and is able to do basic aggregates of the metrics collected. With CloudWatch you can also trigger alerts based on gathered metrics. CloudWatch is one of the most integrated solutions in AWS, almost all of the other services natively send metrics to CloudWatch automatically, including the NLB. CloudWatch alarms are also well connected and are able to trigger a number of actions across the platform. Let's look at how to monitor the AWS NLB with CloudWatch.

Amazon automatically pushes metrics for the majority of their services to CloudWatch, and also allows you to build your own metric collectors for any metric you may want to collect. The Network Load Balancer sends a number of metrics to CloudWatch. These metrics about the NLB include NewFlowCount and ActiveFlowCount, which shows the number of new and active connections the NLB is processing. There are metrics around how many connection resets happen, and there's a metric for each party involved so you can track which party is responsible, TCP_Client_Reset_Count, TCP_ELB_Reset_Count, TCP_Target_Reset_Count. The NLB also tracks how many bytes have been pro-

cessed, and how many load balancer compute units (LCUs) are being consumed. Lastly the NLB will produce CloudWatch metrics on how many healthy and unhealthy targets are behind the load balancer.

CloudWatch enables you to automatically take action based on your metrics. CloudWatch alarms define a threshold on metric statistics. When a metric breaches the alarm threshold, the alarm triggers an action. There are a number of actions that an alarm can configure, such as posting to a Simple Notification Service (SNS) Topic, triggering an Auto Scaling Action, or an EC2 Action. With a SNS topic you can notify other services such as triggering an AWS Lambda function, sending an email, or posting to an HTTP(S) endpoint. With Auto Scaling Actions, you can trigger an Auto Scaling Group to scale up or down based on predefined scaling policies. The EC2 Actions are specific to EC2 Instances with special automation like rebooting or stopping then starting the instance again, which will move the instance to a new physical host. With these actions you have all the integration power to automate reactions to the metrics you collect with CloudWatch, making it a great choice for monitoring your load balancing solution.

Monitoring NGINX

As an application's ingress controller, NGINX provides insights into how your application is performing, who is using it, and how. NGINX provides embedded variables that can be used to log important information; aggregating these logs to a centralized logging service can give you a holistic view of an application. You can also use statsd and collectd plug-ins to ship metrics about NGINX to your monitoring platforms. These metrics are able to tell you more about your network layer, such as the number of connections per second. You can find hypervisor-level metrics for EC2 instances or Elastic Container Service, ECS, containers in CloudWatch collected and aggregated automatically.

By logging appropriately, you can gain not just insight into low-level performance metrics, but high-level context that can aid your performance engineering team. This is a helpful explanation of why log aggregation is important. With a big picture view you can also see patterns to better help you focus your efforts on problem areas, such as attack vectors, failing or slow requests, and how changes affected your service. Some of the important embedded variables that provide this insight are:

- Date and time of the request
- Request URI
- Request method
- Request time in milliseconds

- Request time spent upstream
- Source IP
- Upstream IP
- Country region and city of origin
- Response code
- Other user identifiers

Using these variables in your logs will enable you to analyze your traffic patterns, aggregating and consolidating the logs can give you the insight you need to make important business decisions.

Metric information is also important for monitoring the use of your NGINX servers and machine-level statistics. When running NGINX on AWS, information that can be collected from the hypervisor is automatically sent to CloudWatch. With your metrics in CloudWatch you're able to utilize the same patterns described before. Metrics collected automatically about EC2 instances by AWS include CPU utilization, network utilization, and disk utilization. Metrics collected about ECS containers are simply CPU and memory utilization. These metrics collected by CloudWatch are able to be used to trigger alarms to auto scale your NGINX layer or notify appropriate parties.

With all this information you can make informed decisions about where to focus your work. After you've made changes you can analyze this data to measure the impact of your change. The insight gained from your traffic and utilization patterns can help you discover information you didn't know about your application, or your users.

Monitoring with Amplify

NGINX Inc. provides a monitoring solution called Amplify. Amplify is a Software as a Service (SaaS) monitoring offering. You run the Amplify client on your NGINX node, and Amplify collects the metrics and displays them in meaningful ways. At the time of writing, Amplify does have a free tier, so you can try it out to see what you think. This section will examine some of the metrics Amplify collects and how they're presented.

The Amplify client collects metrics about the operating system, the NGINX service, and other services such as php-fpm and MySQL. Information gathered about the system the Amplify client is running on ranges from CPU and memory, to disk and network usage. The NGINX service provides even more metrics that includes information about how many connections and requests are being handled and their states. The Amplify service also gathers information about HTTP requests and responses served by the NGINX process in detail. Upstream

requests and responses are also monitored and kept separate from the client-side metrics. Amplify also includes information about NGINX cache. NGINX Plus enables a number of other metrics specifically about NGINX Plus features.

For php-fpm Amplify collects metrics on the number of connections, the request processing queue, and the child processes. For MySQL, Amplify collects metrics about connections, number of queries total and by type, innodb buffer stats, and process thread information.

Amplify displays all this information to you through their user interface. The information is displayed as graphs of the metrics over time. You can view the raw graphs themselves, or build your own dashboards with the most important information to you all in one place. The interface is also able to provide an inventory of your servers reporting to Amplify. Amplify is equipped with a configuration analyzer that can make suggestions about how to optimize your NGINX configuration. Amplify tops this all off with the ability to alert based on user-defined rules and thresholds, providing a single interface that addresses all of your monitoring needs for an NGINX or NGINX Plus deployment.

Further Reading

- Amazon CloudWatch
- NGINX Amplify

CHAPTER 7

Scaling and Security

Scaling your application has a lot to do with the way it's delivered and configured. In this chapter you will learn common practices for maintaining order in your environments, and getting the most out of the machines you do run. You will learn about infrastructure and configuration management as code, the NGINX Controller platform, as well as the value of caching and Content Delivery Networks. Finally you will learn how to secure your application from application-level attacks.

Managing Cloud with Infrastructure and Configuration Management

When working with a layer 7 load balancer to deliver an application, you must have a way to manage the configuration. When working in the cloud, with auto scaling applications and load balancers, you will need a way for these services to be provisioned and configured automatically. Serving those needs is configuration and infrastructure management. Configuration management is a process for ensuring a system's configuration is reproducible, and infrastructure management is a process of ensuring infrastructure is reproducible.

Infrastructure and configuration management tools come in the form of declarative state definitions. With these tools you're able to manage your infrastructure and system configurations as code. Just as with applications code it's best practice to keep this code in source control and version it along with your application release cycle. In doing so you're able to reproduce the same environment, from infrastructure, to configuration, and application, which saves you time and effort in not having to reproduce an environment by hand. This process also ensures that what you tested is what you're deploying to production. Reproducible con-

figuration is also necessary for auto scaling as the machines that auto scaling provisions need to have uniform configuration.

The cloud-provided load balancers are provided as a service, their configurations are set as attributes to the cloud resource. These load balancers are configured with infrastructure management. Most of the big cloud providers have means of declarative infrastructure management like CloudFormation which was covered in Chapter 5. There's a third-party tool named Terraform that aims to serve the majority of the cloud providers in a single domain-specific language. The tool you use for declarative infrastructure management in the cloud matters less than the practice itself.

There are a number of notable configuration management tools, such as Ansible, Chef, Puppet, and SaltStack. All of the tools use a declarative state to define system configurations, such as package installs, templated configuration files, services, and support running commands. You can use these tools to version your system configuration, and define system state for your machines. It's common to use configuration management to build bootable machine images, provision machines at launch, or some mix of a partially baked image. These tools enable you to have consistent configurations throughout environments for your layer 7 load balancers and application servers.

By treating your infrastructure and system configuration as code, you're able to follow a full software development lifecycle and adapt all the best practices and procedures used on the application. When infrastructure and system configuration follow those same processes you gain the same advantages: versioning, rollbacks, feature-based branching, code review processes, and the like. These practices produce stronger confidence in deployments, strengthen relations between development and operations teams, and enable methodical testing on all accounts.

NGINX Management with NGINX Controller

NGINX Controller provides a real-time centralized management console to visualize and control your NGINX Plus load balancers and NGINX Unit application servers. With NGINX Controller you can push configurations to NGINX Plus and NGINX Unit servers from a beautiful web-based GUI. Monitoring is also built in via NGINX Amplify, mentioned in Chapter 6.

Managing traffic flow through NGINX Plus servers with NGINX Controller is as easy as point and click, but allows for full featured NGINX configurations directly from the interface. NGINX Controller provides the real-time monitoring insight of NGINX Amplify, so you can directly see the impact of your configuration changes right in the same interface. Built-in authentication and role-based access allows you to enable your teams to only see and change configurations

that are relevant to them. NGINX Controller also allows you to program deployments like canary releases, switching between two different production versions, and rolling back to a prior version. You can then use these deployment mechanisms with a button click.

Visualizing your configuration across multiple environments can help show you commonalities and inconsistencies in your deployment. From this view and ability to swap configuration out in real time on machines, your idea of server and configuration separate enabling your focus to be on the configuration and allowing NGINX Controller to take care of how it gets to the server.

Caching Content and Content Distribution Networks

Caching provides a way to preserve and reserve a response without having to make the full request. Caching can dramatically lower the response time your end users see, and reduce the amount of processing your application needs to do. Most proxies have caching capabilities, including NGINX. A request is cached based on a cache key which is composed of attributes describing the request. When these cache layers are geographically distributed and closer to the end user, they're referred to as Content Delivery Networks or CDNs. By putting your cache layer closer to your user you lower the latency between your user and the application. In this section I'll explain some options for caching and CDNs.

Cloud vendors like AWS provide CDNs native to their platform. AWS's offering is called CloudFront. CloudFront is a globally distributed Content Delivery Network fully integrated into the AWS Cloud. You point your users at the CDN through DNS, their DNS request will route them to the closest cache layer via latency-based routing. From there, CloudFront proxies the request to the application origin, caching the response in order to quickly serve the next user who requests that content. CloudFront is full featured and integrates with the Amazon Web Application Firewall (WAF) to secure your application from client-side attacks.

In some cases it makes more sense to build your own caching layer. These cases typically have to do with cost, targeted performance, or control. You can build your own caching layer with proxies like NGINX. You can distribute your custom-built cache layers to be closer to your end users, building your own CDN. NGINX and NGINX Plus have a rich feature set that makes them an ideal service to build your own cache layer on top of. To learn more about caching with NGINX, I recommend the NGINX ebook, *High-Performance Caching with NGINX and NGINX Plus*.

Whether you build your own or use a prebuilt service offering, caching your content and application responses will increase your customer's satisfaction. The

faster your application operates the better. With a cache layer you're able to lower the response time, the latency, and processing power needed.

Web Application Firewall with ModSecurity 3.0

A web application can be more of a liability than an asset if not secured properly. To protect your web application you can use a Web Application Firewall (WAF). The most common open source WAF is ModSecurity, produced by SpiderLabs. The latest ModSecurity version has added native support for NGINX and NGINX Plus, which allows you to natively take advantage of ModSecurity features to identify and prevent web application attacks directly in NGINX.

The ModSecurity library and NGINX connector module can be compiled and added into your NGINX binary or loaded dynamically. This module provides the ability to run a ModSecurity rule set over each request. ModSecurity rule sets are able to decipher common web application attacks, such as SQL injection, cross-site scripting attacks, vulnerabilities for many application languages, and utilize IP reputation lists. You can build your own ModSecurity rule sets, utilize community-maintained rule sets, or license commercial rule sets from trusted providers.

By identifying and blocking malicious requests from getting to your application, ModSecurity is a great addition to your NGINX Application Delivery Controller. An in-line WAF enables top-grade security in the application layer, where some of the nastiest attacks happen. When you prioritize application security, your application can truly be an asset, keeping your brand in high regard and out of the latest breach headlines.

Further Reading

- Ansible
- Chef
- Puppet
- SaltStack
- CloudFormation
- Terraform
- NGINX-Controller
- High-Performance Caching with NGINX and NGINX Plus
- NGINX-WAF

CHAPTER 8
Conclusion

You should now have the information you need to design a resilient load balancing solution for your cloud application. The approaches, patterns, and examples described in this book will help guide you whether you plan to use NGINX and AWS or another load balancer and cloud provider combination.

As you go forth to build your cloud load balancing solution, keep in mind the importance of automatic registration, deregistration, health checking, and the ability to scale your load balancer horizontally. Think about the likelihood of your solution needing to exist outside of your cloud vendor, and determine the value of your solution being portable to other providers. Consider your application needs for session persistence and examine if now is the right time to centralize session state to alleviate this need.

If you have not already, try out the Amazon Quick Start for NGINX Plus to get a feel for the AWS environment and see if NGINX or NGINX Plus is a good fit for your solution. Take note of the value provided by CloudFormation and Configuration Management that enables you to build up the Quick Start without any manual configuration. Remind yourself that Infrastructure and Configuration management not only provides you a repeatable deployment but also the ability to reliably test and build quality gates for your systems and security.

Consider your monitoring and logging plan and how your load balancing solution will integrate with that plan to give you the insight you need to maintain, support, and scale the application.

Lastly, remember that there are many ways to load balance in the cloud and that it's up to you to choose the most fitting path for your organization and application.

About the Author

Derek DeJonghe has had a lifelong passion for technology. His in-depth background and experience in web development, system administration, and networking give him a well-rounded understanding of modern web architecture. Leading a team of cloud architects and solution engineers, to producing self-healing, auto scaling infrastructure for numerous different applications, Derek specializes in cloud migrations and operations of all sizes. While designing, building, and maintaining highly available applications for clients, Derek often engages in an embedded consulting role for larger organizations as they embark on their journey to the cloud. Derek and his team are on the forefront of a technology tidal wave and are engineering cloud best practices every day. With a proven track record for resilient cloud architecture, Derek helps RightBrain Networks be one of the strongest cloud consulting agencies and managed service providers in partnership with AWS today.