CISCO ACE TO NGINX: Migration Guide

CISCO ACE TO NGINX:

Migration Guide

by Faisal Memon

Why Replace Cisco ACE with NGINX Plus?

Flexibility – You can deploy NGINX Plus anywhere. In your private datacenter, AWS, Microsoft Azure, Google Cloud, and almost any environment.

Cost savings – NGINX Plus is so efficient and lightweight that you can run it on off-the-shelf servers costing under a $1,000. That's more than 85% savings over a replacement hardware load balancer such as F5 BIG-IP or Citrix NetScaler.

Open source – NGINX Plus is based on NGINX open source, which is backed by an enthusiastic community of over 450 million users. There are thousands of community-contributed articles to help you on your journey.

We're spending substantially less time on website management than with Cisco ACE. With NGINX Plus, we're able to spin things up and test them ridiculously fast.

–Drew Turner, Web Operations Manager at MidwayUSA

Learn more at nginx.com

Table of Contents

1 Introduction

"In a future that includes competition from open source, we can expect that the eventual destiny of any software technology will be to either die or become part of the open infrastructure itself."

–Eric S. Raymond, *The Cathedral and the Bazaar*, 1997

Cisco entered the hardware load balancer market with an add-on "Application Control Engine" (ACE) module for its Catalyst 6500-series switches, the Cisco ACE30. Soon after they released the Cisco ACE 4710 as a stand-alone hardware appliance. The ACE line was doing well for Cisco with a $50 million run rate. But in 2012, Cisco abruptly announced end-of-life for the CIsco ACE.

Even during the Cisco ACE''s heyday, a fundamental transformation was already underway in the data center: enterprises were starting to move away from proprietary hardware appliances and towards open source software. This was driven partly by the increasingly active role of developers in delivering the applications they write – they in particular favor lighter, more agile, software-defined, and open source solutions that can be installed on standard x86 servers. Cisco no longer felt it could compete in this changing market.

NGINX, Inc. has many customers that have replaced Cisco ACE with NGINX Plus, such as Midway USA, Genomics Institute of the Novartis Research Foundation, and Cleveland State University. The final chapter of this migration guide is a case study of Midway USA's journey from Cisco ACE to NGINX Plus.

The remainder of the migration guide explains how to replace Cisco ACE hardware appliances using NGINX and NGINX Plus. NGINX is an open source load balancer, content cache, and web server. NGINX Plus is our commercial and fully supported product, with exclusive features such as HTTP health checks, session persistence (what Cisco ACE calls HTTP Cookie stickiness), and high availability (HA). Because of its enhanced features, NGINX Plus is usually the better choice when replacing Cisco ACE.

Note: For ease of reading, we refer to NGINX when the instructions are applicable to both NGINX and NGINX Plus. NGINX Plus exclusive functionality will be noted.

Why Software Load Balancing?

Hardware-based load balancers – often called application delivery controllers (ADCs) – were introduced twenty years ago as a way to get the most out of high-cost, first-generation web servers. But application and website delivery has changed dramatically from the days when it took racks of specialized devices to optimize performance.

Now it's the hardware load balancers themselves that are the high-cost, antiquated devices. The companies succeeding at web scale today use load balancers powered by low cost, flexible, and easily configured software, running on either commodity hardware or virtualized instances.

IT buyers in enterprises are following the example of the web-scale pioneers, and architectures are changing. Hardware ADC product sales are flat or down, and the device makers now face management and staff turnover and uncertainty about their future. Every contract renewal date, every increase in costs, and every innovation in alternative software and cloud solutions sees more customers migrating from legacy hardware solutions to the latest generation of software ADCs.

Now, the industry is at a crossroads. Hardware ADC makers are introducing a new generation of more complex, more expensive hardware ADCs, while trying to hedge their bets with bloated adaptations of their firmware to "software deployable" instances. Customers have a choice: either continue to be locked into antiquated technology and one-sided business relationships, or move to software-based application delivery controllers, such as NGINX, and immediately reduce costs, increase performance, and unlock greater flexibility and control.

Here are five reasons to move from traditional hardware ADCs to software solutions:

1. Dramatically reduce costs without sacrificing features or performance. You can save roughly 85% for the same price/performance with NGINX Plus vs. F5 BIG-IP or Citrix NetScaler.

2. Enable modern development and delivery frameworks, like DevOps. As applications move to continuous integration/continuous delivery (CI/CD), and as teams restructure to make rapid deployments possible, waiting days or weeks for an ops team to reconfigure a hardware ADC is unimaginable. Only software provides the rapid configuration, ease of flexibility, and application-level control that DevOps requires.

3. Deploy one ADC solution everywhere. As software, NGINX Plus works the same way on all platforms – in VMs or containers, on premises, and on public, private, or hybrid cloud – making deployment flexible and easy.

4. Adapt quickly to changing demands on your applications. No waiting for special hardware, installation, and configuration when traffic is rising. It takes just minutes for Installation and configuration before you can immediately scale up or scale out your applications to respond in real time to the demands on them.

5. Eliminate artificial or contract-driven constraints on performance. Hardware ADCs cap throughput based on your contracted costs, then hit you with hefty upgrade fees when traffic breaks through that cap. Software-based solutions have no artificial limits – simply upgrade the underlying hardware or virtualized compute power to unlock greater performance. Say goodbye to hefty fees for traffic increases.

Why Open Source?

Whether or not we realize it, we interact with open source software on a daily basis. Apple MacOS is based on the open source FreeBSD. Amazon Elastic Compute Cloud (EC2) is built on the open source Xen hypervisor. Even newer Cisco products (such as the Cisco Nexus line of switches) use Linux underneath the hood.

The technological innovation we benefit from today is only possible because of open source. People freely sharing ideas and code has enabled us to move much faster than if we all stayed in our silos not collaborating or sharing. In the next 20 years, it will be nearly impossible to develop large-scale software or platforms without using open source software.

Open source software is free and easily attainable, providing great benefits to the business. Rather than waiting weeks to procure a custom hardware appliance, anyone can download open source software in minutes.

The best part of open source is the community. NGINX, for example, is used by 458 million websites according to the January 2017 Netcraft web server survey. As a result there is a community around NGINX, providing help, guidance, and tip and tricks to other users. The only way to get that reach is with open source.

Brief History of NGINX

> *"When I started NGINX, I focused on a very specific problem – how to handle more customers per a single server."*
>
> –Igor Sysoev, NGINX creator

The open source NGINX software was created by Igor Sysoev as a side project while he was working as a sysadmin at Rambler, a Russian equivalent of Yahoo!. While at Rambler, Igor was asked to look into enabling the Apache HTTP servers to better handle the influx of traffic the company was receiving.

Apache HTTP Server is a web server, a website component that is responsible for delivering static assets, such as images, to users. While looking for ways to improve Apache's performance, Igor found himself blocked by several inherent

design choices that hampered Apache's ability to handle 10,000 simultaneous users, commonly known as the C10K problem. In the spring of 2002 Igor started developing NGINX with an event-driven architecture that addressed the shortcomings in Apache.

On October 4th, 2004, the anniversary of the launch of Sputnik, the first space satellite, Igor publicly released the source code of NGINX for free. From there NGINX quickly gained in popularity amongst web developers who immediately benefit from its improved performance over Apache.

Seven years later, Igor and co-founders Andrew Alexeev and Maxim Konovalov created NGINX, Inc. and in 2013 the first version of NGINX Plus, the commercial offering, was released.

Today NGINX, Inc is a global company with offices in San Francisco, Moscow, Singapore, and Cork. It is a venture capitalist (VC) funded by firms such as Telstra, NEA, and eVentures.

2 Prepare Your Server

"It is said that the only things certain in life are death and taxes. For those of us in the IT industry, we can add one more to the list: commoditization."

–Ian Murdock, Creator of Debian Linux

One of the biggest benefits of a hardware appliance is that all the work of setting up a server is done for you. When using open source software, or software solutions in general, you have to prepare the server yourself. For NGINX or NGINX Plus, this includes purchasing a server of the right size, installing Linux, and finally installing NGINX or NGINX Plus.

In this chapter we walk you step-by-step through the process to make it as easy as possible By the end of the chapter you will know how to prepare a server and may even prefer it over the turnkey hardware appliance, because there is much more room for customization

Purchasing a Commodity Server

There are many benefits to commodity servers. For example, if something fails, it is much easier and quicker to bring up a new x86 server than to procure a new hardware appliance. It's also easier to right-size the solution for your application needs, rather than buying a hardware appliance that is less customizable.

In this section we will cover the steps need to purchase an appropriately sized server.

Cisco ACE Performance Profile

Cisco ACE has a 3.4 GHz Intel Pentium 4 processor, according to the official documentation. Cisco uses software-based restrictions to create models of the ACE 4710 hardware appliance at various levels of performance. With the top-end Cisco ACE 4710 you get:

- 4 Gbps throughput
- 7,500 SSL transactions per second (TPS)
- 2 Gbps compression throughput
- 20 virtual contexts

The Cisco documentation does not specify which SSL algorithm was used to attain the 7,500 SSL TPS. The strongest encryption algorithm supported by Cisco ACE SSL hardware acceleration is RSA_WITH_AES_256_CBC_SHA, which falls short of modern SSL standards.

Equivalent x86 Server

NGINX runs on commodity x86 servers that you can purchase from Dell, HPE, Lenovo, SuperMicro and other vendors. These servers do not have built-in hardware SSL acceleration, but thanks to the drastic increase in compute power due to Moore's law, a low-end x86 server can easily outperform even the high-end Cisco ACE.

Perfect forward secrecy

Modern SSL standards mandate perfect forward secrecy (PFS), which ensures that encrypted traffic captured at a certain time (in a man-in-the-middle attack, say) can't be decrypted later (at the hijacker's leisure), even if the private key is compromised. PFS is recommended for maximum protection of user privacy in the current security climate.

PFS is more computationally expensive and so reduces TPS. As a result most vendors do not publish TPS data for PFS; keep this in mind when comparing specifications from vendors.

Hardware specifications	Expected performance	Cost
• 2 CPU cores @ 3.7 GHz • 4 GB RAM • 2x10 Gbe NIC • 500 GB HDD	• 90,000 requests per second (RPS) • 1,500/4,700 RSA/ECC SSL TPS[1, 2] • 9 Gbps throughput	$1,200

1. RSA: 2048 bit, ECDHE-RSA-AES256-GCM-SHA384; ECC: ECC 256 bit, ECDHE-ECDSA-AES256-GCM-SHA384
2. Using OpenSSL 1.0.2

This server not only matches, but exceeds, the performance of the Cisco ACE. Though the SSL TPS numbers are lower than for the Cisco ACE, it's important to note that they are for SSL with PFS, which is significantly more secure but reduces performance (see "Perfect forward secrecy" sidebar). The algorithm used for the Cisco ACE number does not include PFS.

While pricing for Cisco ACE is no longer available, hardware from F5 or Citrix with similar specs starts at $18,000, 15 times more than the commodity server. (Hardware vendors also typically charge up to 18% of the list price as an annual support and maintenance fee. There is no corresponding fee for commodity hardware, though NGINX Plus has an annual subscription fee.)

Note: NGINX is also supported on ARM and Power8 servers for some types of Linux.

Installing Linux

If your experience is with hardware appliances and you've have never used Linux before, installing it may seem daunting. It's not easy to learn a new operating system (OS). But if you've figured out Cisco IOS command syntax, you're more than capable of learning Linux! Plus, Linux is becoming the de facto industry-standard OS in modern IT environments, so learning it is mandatory for anyone continuing to work in the industry.

In this section we explain how to pick a Linux distribution and install it.

Choosing the Right Distro

There are many Linux distributions (distros) and versions of each, which may add to the confusion. Of the many Linux distros, we recommend either of the following:

- Ubuntu Server 16.04 LTS or later
- CentOS or Red Hat Enterprise Linux (RHEL) 7.4 or later (CentOS is the community edition of RHEL)

Note: The remainder of this migration guide will assume you are using one of these operating systems.

These distributions all come with version 1.0.2 of the OpenSSL library. OpenSSL is the software that handles SSL/TLS operations in Linux. Version 1.0.2 performs 2 to 3 times better than version 1.0.1 (found in older Linux distros). OpenSSL 1.0.2 is also a requirement for HTTP/2, the latest version of the HTTP standard which also improves performance.

NGINX supports Ubuntu Server 16.04 LTS on x86, ARM, and Power8 servers, and CentOS/RHEL 7.4+ on x86 and Power8 servers.

Professional support is available for both Ubuntu Server and RHEL, from Canonical and Red Hat respectively.

Installing Linux

Before installing Linux, you need two things:

- A USB memory stick with 4 GB or more
- A server with connectivity to a network with DHCP and Internet access, both required for the Linux installer to retrieve packages over the Internet.

After you meet the prerequisites, there are just five steps:

1. Download the .iso file for CentOS 7.4+, RHEL 7.4+, or Ubuntu Server 16.04 LTS.

   You can download CentOS and Ubuntu Server for free; for RHEL you must first purchase a subscription or start an evaluation.

2. Create a bootable USB stick, the easiest way to install Linux. Ubuntu provides instructions for Mac and Windows, but they apply to CentOS and RHEL as well – just substitute the appropriate **.iso** filename.

3. Boot up the server with the bootable USB stick. This is the default boot process on most servers but if that is not the case, check the boot order settings in your BIOS.

4. Follow the wizard to install Linux. This is normally straightforward, requiring you only to set the date, time, and time zone. Be sure not to install the GUI if that option is offered.

5. Configure a static IP address. There are separate instructions below for CentOS/RHEL and Ubuntu Server.

Configuring a Static IP Address on CentOS and RHEL

To configure a static IP address on CentOS and RHEL we will use the Network Manager.

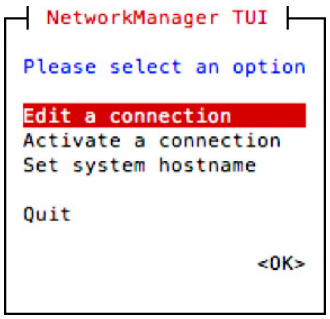
1. Use `nmcli` to list the available network interfaces:

```
$ nmcli
ens33: connected to ens33
    "Intel 82545EM Gigabit Ethernet Controller (Copper) (PRO/1000 MT
    Single Port Adapter)"
    ethernet (e1000), 00:0C:29:18:91:6A, hw, mtu 1500
    ip4 default
    inet4 192.168.179.232/24
    inet6 fe80::4978:2423:e3e6:536f/64
```

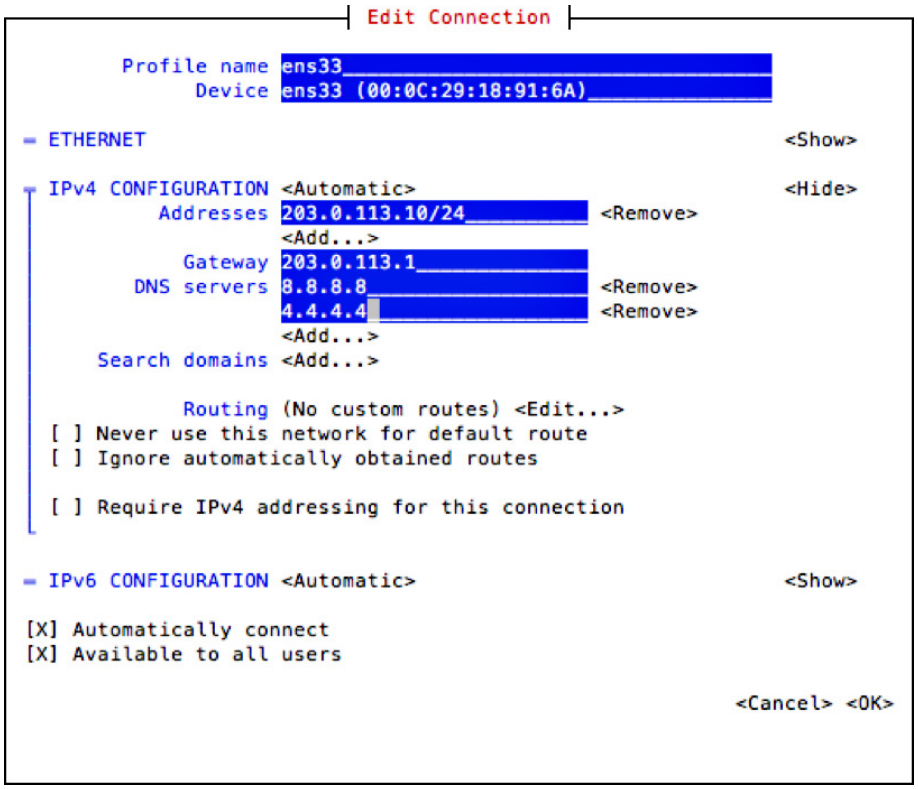
The `inet4` field lists the current IP Address of the interface. The name of the interface in this example is `ens33`. The value varies by system and is based on multiple factors such as the PCI Express hotplug slot index number. If there are multiple interfaces they will all be listed here.

2. Bring up the Network Manager test-based user interface (TUI):

```
$ sudo nmtui
```



3. Select **Edit** a connection and then select the appropriate interface in the list.
4. In the **Edit** Connection menu, change IPv4 Configuration from **Automatic** to **Manual** and the press Enter on **<Show>** to show the current IP settings.
5. Modify the IP settings for your environment



Useful Linux commands:

`ls -lF` – List all files including file sizes

`rm` *`file`* – Remove a file

`cp` *`file1 file2`* – Copy file1 to file2

`cat` *`file`* – View the contents of a file

`mkdir` *`directory`* – Create a new directory

`rm -rf` *`directory`* – Forcefully remove a directory and all contents

`pwd` – Display the present working directory

`top` – Display top processes

`ps -ef` – Display all currently running processes on the system, including process id (PID)

`ps -ef | grep` *`processname`* – Display process info for processname

`kill pid` – kill process with process ID of *pid*

6. Press <OK> to save the settings, and the press Esc twice to exit out of nmtui.
7. Restart the networking service for the changes to take effect.

```
$ sudo systemctl restart network
```

Configuring a Static IP Address on Ubuntu Server Linux

To configure a static IP address on Ubuntu Server Linux, you use a text editor to edit the **/etc/network/interfaces** file, changing the default DHCP setting to add some information about the static IP address that you want to configure. Here we're using the nano text editor.

1. Open **/etc/network/interfaces** in the text editor.

```
$ sudo nano /etc/network/interfaces
```

2. Search for "**# The primary network interface**". The two lines after it look something like this:

```
# The primary network interface
auto ens33
iface ens33 inet dhcp
```

   The name of the interface in this example is **ens33**. The value varies by system and is based on multiple factors such as the PCI Express hotplug slot index number.

3. Comment out the line that ends with dhcp by putting a **#** in front of it, then add the following lines so that the complete section looks like this:

```
# The primary network interface auto ens33
# iface ens33 inet dhcp
iface ens33 inet static
    address 203.0.113.10
    netmask 255.255.255.0
    gateway 203.0.113.1/24

dns-nameservers 8.8.8.8 8.8.4.4
```

   Replace the two instances of the server's IP address (203.0.113.1) and the IP addresses for the DNS name server (8.8.8.8 and 8.8.4.4) with the appropriate values for your server. This example refers to Google's public DNS servers.

4. Save and close the file. With nano, press Ctrl+x to exit and enter y to save. Press Enter again to overwrite the current file.
5. Normally you do not need to reboot to update network settings. You can just run this command to restart the networking service:

```
$ sudo /etc/init.d/networking restart
```

   If that does not work, restart Linux:

```
$ sudo restart now
```

Installing NGINX or NGINX Plus

Now that you've installed Linux, the next step is to install NGINX or NGINX Plus. The installation steps differ for NGINX and NGINX Plus, and further vary depending on the operating system you are using. We're providing instructions for both NGINX and NGINX Plus on both reference OSs.

NGINX Plus is available at per-instance subscription prices for small deployments and at special package rates for larger deployments. For details, see our pricing page. You can also try NGINX Plus for free for 30 days or buy NGINX Plus online.

Once your subscription or free trial begins, NGINX, Inc. will send and instructions for logging in at the NGINX Plus Customer Portal.

If you're not sure whether NGINX or NGINX Plus makes more sense for you, see the feature comparison in Appendix A.

Installing Open Source NGINX on CentOS and RHEL

1. Add the official NGINX repository to your `yum` repositories. Run the following command:

```
$ sudo echo \
"[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/OS/7/\$basearch/
gpgcheck=0
enabled=1" > /etc/yum.repos.d/nginx.repo
```

   Replace *`OS`* with `centos` or `rhel` as appropriate. If you are using CentOS/RHEL 6.5+, change 7 to 6.

2. Install NGINX:

```
$ sudo yum update && sudo yum install nginx
```

3. Start NGINX and set it to start automatically on reboot:

```
$ sudo systemctl enable nginx
$ sudo systemctl start nginx
```

4. Enable incoming traffic to port 80, because by default CentOS and RHEL block all incoming connections. If NGINX will be listening on other ports (for example, on port 443 for HTTPS), repeat the first **`firewall`** command for each one:

```
$ sudo firewall-cmd --permanent --zone=public --add-port=80/tcp
$ sudo firewall-cmd --reload
```

Installing Open Source NGINX on Ubuntu Server Linux

Add the NGINX signing key to the **`apt`** program keyring:

```
$ wget -qO - https://nginx.org/keys/nginx_signing.key | sudo
apt-key add -
```

2. Add the official NGINX repository to your apt repositories. Run the following command:

```
$ sudo echo \
“deb http://nginx.org/packages/mainline/ubuntu/ codename nginx
deb-src http://nginx.org/packages/mainline/ubuntu/ codename nginx” \
| sudo tee /etc/apt/sources.list.d/nginx.list
```

Replace ***`codename`*** with **`xenial`** for Ubuntu 16.04 or **`zesty`** for Ubuntu 17.04.

3. Install NGINX:

```
$ sudo apt-get update && sudo apt-get install -y nginx
```

4. Start NGINX (it will automatically start on reboot):

```
$ sudo /etc/init.d/nginx start
```

Installing NGINX Plus on CentOS and RHEL

1. Create the /etc/ssl/nginx directory:

```
$ sudo mkdir /etc/ssl/nginx
$ cd /etc/ssl/nginx
```

2. Log in to NGINX Plus Customer Portal.

3. Download your version of *nginx-repo.crt* and *nginx-repo.key* files from NGINX Plus Customer Portal and copy the files to **/etc/ssl/nginx/** directory:

```
$ sudo cp nginx-repo.crt /etc/ssl/nginx/
$ sudo cp nginx-repo.key /etc/ssl/nginx/
```

4. Install **ca-certificates**:

```
$ sudo yum install ca-certificates
```

5. Download the **nginx-plus-repo** file and copy it to the **/etc/yum.repos.d/** directory:

```
$ sudo wget -P /etc/yum.repos.d \
```

https://cs.nginx.com/static/files/nginx-plus-7.4.repo

For CentOS/RHEL 7.0-7.3:

```
$ sudo wget -P /etc/yum.repos.d \
```

https://cs.nginx.com/static/files/nginx-plus-7.repo

For CentOS/RHEL 6:

```
$ sudo wget -P /etc/yum.repos.d \
```

https://cs.nginx.com/static/files/nginx-plus-6.repo

6. Install NGINX Plus:

```
$ sudo yum install nginx-plus
```

7. Start NGINX and set it to start automatically on reboot:

```
$ sudo systemctl enable nginx
$ sudo systemctl start nginx
```

8. Enable incoming traffic to port 80, because by default CentOS and RHEL block all incoming connections. If NGINX will be listening on other ports (for example, on port 443 for HTTPS), repeat the first **`firewall`** command for each one.

```
$ sudo firewall-cmd --permanent --zone=public --add-port=80/tcp
$ sudo firewall-cmd --reload
```

Installing NGINX Plus on Ubuntu Server Linux

1. Create the /etc/ssl/nginx directory:

```
$ sudo mkdir /etc/ssl/nginx
$ cd /etc/ssl/nginx
```

2. Log in to NGINX Plus Customer Portal.

3. Download your version of *nginx-repo.crt* and *nginx-repo.key* files from NGINX Plus Customer Portal and copy the files to **/etc/ssl/nginx/** directory:

```
$ sudo cp nginx-repo.crt /etc/ssl/nginx/
$ sudo cp nginx-repo.key /etc/ssl/nginx/
```

4. Download the NGINX signing key from nginx.org and add it:

```
$ wget -qO - https://nginx.org/keys/nginx_signing.key | \
sudo apt-key add -
```

5. Install **apt-utils** package and install NGINX Plus repository:

```
$ sudo apt-get install apt-transport-https lsb-release \
ca-certificates
$ printf “deb https://plus-pkgs.nginx.com/ubuntu ’lsb_release -cs’ \
nginx-plus\n” | sudo tee /etc/apt/sources.list.d/nginx-plus.list
```

6. Download the ***90nginx*** file to **/etc/apt/apt.conf.d**:

```
$ sudo wget -qO /etc/apt/apt.conf.d/90nginx \
```

https://cs.nginx.com/static/files/90nginx

7. Install NGINX Plus:

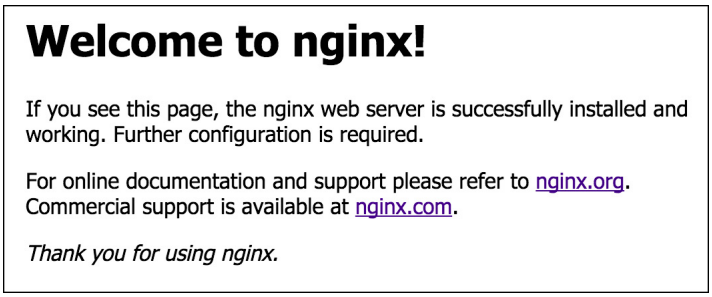
```
$ sudo apt-get update & sudo apt-get install -y nginx-plus
```

8. Start NGINX (it will automatically start on reboot):

```
$ sudo /etc/init.d/nginx start
```

Verifying Installation

For both NGINX and NGINX Plus, you can verify successful installation by navigating in a browser to the hostname or IP address of the NGINX or NGINX Plus server. The "Welcome to nginx!" page confirms that the server is running:



If you can access the "Welcome to nginx!" page, you've sucessfully installed NGINX

3 Convert Cisco ACE Configuration to NGINX

"I wanted people to use it,
so I made it open source."

–Igor Sysoev, creator of NGINX

Converting your Cisco ACE configuration to NGINX is probably the most time-consuming part of a migration, because the configuration methods are quite different. Cisco ACE uses the Cisco IOS command-line interface, which requires entering commands and doing a "`wr mem`" to save the set of commands as a configuration. NGINX uses a plain-text configuration file with C-like syntax.

In this chapter we show how to convert the configuration of common Cisco ACE features to the equivalent in NGINX. As with any translation, an exact equivalent is not always possible. Also, some Cisco ACE configuration requirements are unnecessarily complex and no longer reflect industry best practices; we note this for relevant cases below.

NGINX Configuration Basics

Here is some basic configuration information for NGINX:

- All NGINX configuration is stored in the **/etc/nginx/conf.d/** folder. With this folder, all relevant configuration file should be stored in a file following this format: **www.example.com.conf**. Replace **www.example.com** with the domain of your site.

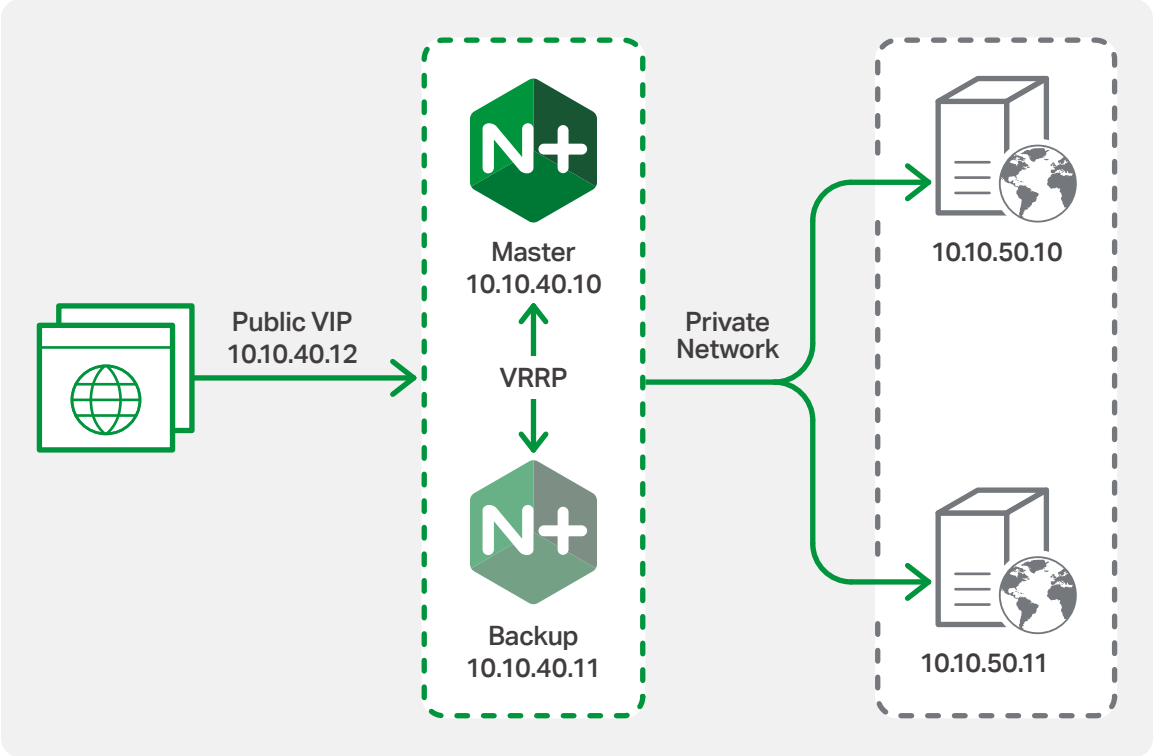
After modifying the configuration, you must reload the nginx process to start using it. We also recommend that you run `nginx -t` to verify the configuration is syntactically correct. Reloading does not cause service disruption:

```
nginx -t && nginx -s reload
```

- The main NGINX configuration file is **/etc/nginx/nginx.conf**. If you use the configuration scheme described above, it's not usually necessary to modify this file.
- NGINX configuration is not order-dependent.

Topology

The configuration examples in this chapter apply to the following topology:



This two-armed topology with NGINX Plus will be used in this chapter

This is a two-armed configuration with separate subnets, one for the site's public-facing IP addresses or virtual IP addresses (VIPs) assigned to the NGINX server, and the other for the actual web or application servers, which Cisco ACE calls the *real servers*. A one-armed configuration with all servers on a single subnet is also possible and quite common.

Creating a Server Farm

A Cisco ACE *server farm* is a list of the IP addresses and port numbers of a set of *real servers* (backend web or application servers) that all provide the same functionality. NGINX refers to this as an *upstream group*.

Cisco ACE

```
rserver host RS_WEB1
    description content server web-one
    ip address 10.10.50.10
    inservice
rserver host RS_WEB2
    description content server web-two
    ip address 10.10.50.11
    inservice
serverfarm host SF_WEB
    rserver RS_WEB1 80
        inservice
    rserver RS_WEB2 80
        inservice
```

NGINX

```
upstream SF_WEB {
    server 10.10.50.10:80; # content server web-one
    server 10.10.50.11:80; # content server web-two
}
```

Explanation of NGINX directives:

- **`upstream`** – Defines the name of the server farm and corresponds to `serverfarm host` in Cisco ACE configuration.
- **`server`** – Defines the IP address:port for each server in the server farm. This directive consolidates the ACE `rserver host` and `ip address` configurations into one line. The optional **`#`** comment can be used in place of the ACE `description` field. In Cisco ACE, each physical server is also assigned a name, but this has no practical purpose and there is no corresponding NGINX directive.

Notes:

- You can omit the port number in the NGINX server directive, in which case NGINX forwards traffic to port 80 on the upstream servers. In Cisco ACE, when the port is omitted the port requested by the client is passed through.
- By default NGINX treats all servers as inservice. To explicitly disable a server, use the down parameter to the server directive:

```
server 10.10.50.10:80 down; # content server web-one
```

Creating a Virtual Server

The virtual server defines the IP address and other necessary parameters for client-side connectivity.

Cisco ACE

```
class-map match-all VS_WEB
    2 match virtual-address 10.10.40.10 tcp eq 80
policy-map type loadbalance first-match PM_LB
    class class-default
        serverfarm SF_WEB
policy-map multi-match PM_MULTI_MATCH
    class VS_WEB
        loadbalance vip inservice
        loadbalance policy PM_LB
```

NGINX

```
server {
    listen 10.10.40.10:80 default_server;
    server_name www.example.com;

    location / {
        proxy_set_header Host $host;
        proxy_pass http://SF_WEB;
    }
}
```

Explanation of NGINX directives:

- **server** – Defines a virtual server. You cannot name a virtual server as is required in Cisco ACE. To help you distinguish easily between NGINX Plus virtual servers, you can define each one in a separate file in the conf.d directory.
- **listen** – Defines the IP address and port on which NGINX listens for client requests. If the IP address is omitted, NGINX binds to all IP addresses available on the system. See the sections: Configuring a Static IP Address on Ubuntu Server Linux or Configuring a Static IP Address on CentOS and RHEL.

- **proxy_pass** – Names the server farm to which NGINX forwards requests that match the location block that encloses this directive. The forward slash (**/**) argument to location used in this example means that NGINX directs requests for all paths beginning with **www.example.com** to the **SF_WEB** server farm

Configuring a Load-Balancing Predictor

Load-balancing algorithms (which Cisco ACE calls *predictors*) determine how the load balancer distributes traffic among the servers in the server farm. Both Cisco ACE and NGINX do round-robin load balancing by default, but also support more complex algorithms that base the load-balancing decision on multiple factors.

Cisco ACE

```
serverfarm host SF_WEB
    predictor leastconns
    rserver RS_WEB1 80
        inservice
```

NGINX

```
upstream SF_WEB {
    server 10.10.50.10:80; # content server web-one
    server 10.10.50.11:80; # content server web-two

    least_conn;
}
```

Explanation
These examples configure the “least connections” algorithm, in which the load balancer sends each request to the server that currently has the lowest number of connections.

Use the following table to translate load-balancing algorithms from Cisco ACE to NGINX:

Cisco Ace	NGINX	Description
`predictor leastconns`	`least_conn`	Choose the server with the lowest number of existing connections
`predictor response`	`least_time` (exclusive to NGINX Plus)	Choose the server with fastest response time
`predictor hash address source`	`hash $remote_addr`	Hash the source IP address and choose the server to which the hash is assigned (establishes session persistence)
`predictor hash url`	`hash $request_uri`	Hash the request URI and choose the server to which the hash is assigned

Notes:

- With the NGINX hash method, you can add the consistent parameter to reduce the amount of hash reassignment that happens when the set of real servers changes. It uses the ketama hashing algorithm.
- Cisco ACE supports many other load-balancing algorithms, but most are now considered obsolete. The four algorithms in NGINX (plus least_time in NGINX Plus) represent current best practice.

Least time load balancing is exclusive to NGINX Plus

Configuring HTTP Cookie Stickiness

For applications that maintain session state on the server itself, it's important to ensure that all requests from a given client go to the same server. Both Cisco ACE and NGINX Plus support this feature, which they call *HTTP cookie stickiness* and *cookie-based session persistence* respectively (NGINX Plus supports other types of session persistence as well).

This type of session persistence is exclusive to NGINX Plus. NGINX Open Source supports a simpler form of session persistence, implemented with the hash or `ip-hash` load-balancing algorithm discussed above.

Cisco ACE

```
sticky http-cookie Cookie1 StickyGroup1
    cookie insert browser-expire
    timeout 3600
    serverfarm SF_WEB
policy-map type loadbalance first-match PM_LB
    class class-default
        sticky-serverfarm STICKYGROUP1
```

NGINX Plus

```
upstream SF_WEB {
    server 10.10.50.10:80; # content server web-one
    server 10.10.50.11:80; # content server web-two

    sticky cookie Cookie1 expires=1h domain=.example.com path=/;
}
```

Explanation of NGINX Plus directive:

- **server** – Defines a virtual server. You cannot name a virtual server as is required in Cisco ACE. To help you distinguish easily between NGINX Plus virtual servers, you can define each one in a separate file in the conf.d directory.
- **sticky cookie** – Defines the sticky cookie. The first parameter names the cookie and corresponds to the first parameter to **sticky http-cookie** in Cisco ACE. The **expires** parameter corresponds to the Cisco ACE **timeout** field. The optional **domain** parameter defines the domain for which the cookie is set, and the optional **path** parameter the request path for which the cookie is set. These parameters do not have direct equivalents in Cisco ACE.

  Cisco ACE uses the second parameter to **sticky http-cookie**, here **StickyGroup1**, to associate the persistence policy with a virtual server. NGINX Plus instead associates the policy with the server farm.

Configuring SSL Termination

With SSL termination, the load balancer terminates SSL/TLS connections from clients and uses unencrypted (clear-text) connections to the real servers. This practice offloads the computational load of encryption and decryption from the real servers, enabling them to concentrate on their actual function.

NGINX can also re-encrypt traffic to the real servers, if needed.

Cisco ACE

```
sticky http-cookie Cookie1 StickyGroup1
    timeout 3600
    serverfarm SF_WEB
ssl-proxy service PS_SSL_TERMINATION
    key cisco-sample-key
    cert cisco-sample-cert
    class-map match-all CM_SSL
    2 match virtual-address 10.10.40.11 tcp eq https

policy-map multi-match PM_MULTI_MATCH
    class VS_WEB
        loadbalance vip inservice
        loadbalance policy PM_LB
    class CM_SSL
        loadbalance vip inservice
        loadbalance policy PM_LB
        ssl-proxy server PS_SSL_TERMINATION
```

NGINX

```
server {

    listen 10.10.40.11:443 ssl default_server;
    server_name www.example.com;

    ssl_certificate cisco-sample-cert;
    ssl_certificate_key cisco-sample-key;

    location / {
        proxy_set_header Host $host;
        proxy_pass http://SF_WEB;
    }
}
```

Explanation of NGINX directives:
NGINX SSL/TLS termination configuration is at the virtual server level.

- **`listen`** – As for unencrypted HTTP traffic, defines the IP address and port on which NGINX listens for client requests; here the **`ssl`** parameter tells it to accept only encrypted traffic over HTTPS. If the IP address is omitted, NGINX binds to all the addresses on the system that Linux is bound to.
- The **`ssl_certificate`** and **`ssl_certificate_key`** directives correspond to the Cisco ACE **`cert`** and **`key`** directives and define the public certificate and private key for encrypted communication with clients. It is recommended to put both in the **/etc/nginx/** directory on your server. If you don't specify a pathname, NGINX looks for the certificate and key in **/etc/nginx**.
- As previously noted, NGINX uses unencrypted connections to the servers in the server farm. To configure end-to-end SSL/TLS, change **`http://`** to **`https://`** in the **`proxy_pass`** directive. NGINX still terminates the encrypted client connection and establishes a separate encrypted connection to the servers in the server farm.

Configuring Health Monitoring Using Health Probes

To detect failures and make reliable load-balancing decisions, you can configure both Cisco ACE and NGINX Plus to monitor the health of real servers by periodically sending health probes, which NGINX Plus calls health checks. A server that does not respond correctly to the probe is considered unhealthy, and the load balancer doesn't sent traffic to it until it becomes healthy again.

This type of active (out-of-band) health check is exclusive to NGINX Plus. Along with NGINX Plus, NGINX Open Source supports passive health checks, which means that it marks a server as unhealthy when it fails to handle an actual client request.

Cisco ACE

```
probe http HTTP_PROBE1
    expect status 200 200

serverfarm host SF_WEB
    predictor leastconns
    probe HTTP_PROBE1
    rserver RS_WEB1 80
        inservice
```

NGINX Plus

```
upstream SF_WEB {
    zone backend 64k;
    server 10.10.50.10:80; # content server web-one
    server 10.10.50.11:80; # content server web-two

    least_conn;
}

server {

    listen 10.10.40.10:80 default_server;
    server_name www.example.com;

    location / {
        proxy_set_header Host $host;
        proxy_pass http://SF_WEB;
        health_check;
    }
}
```

Explanation of NGINX Plus directives:

- **zone** – Creates a shared memory zone for storing information about health checks (mandatory).
- **health_check** – Configures health checks to the servers in the SF_WEB server farm, with the default parameters: NGINX Plus sends a request for **/** to each server every five seconds. When a server fails one health check, it is marked unhealthy and receives no traffic until it responds correctly to one subsequent health check.

Configuring Active-Passive Redundancy

Cisco ACE and NGINX Plus handle redundancy and fault tolerance very differently, starting with the terminology: the term *high availability* (HA) covers both in NGINX Plus. NGINX Plus does not synchronize connection state across the cluster of load balancers as Cisco ACE does; connection state is lost on failover. However, synchronizing connection status is not vital for HTTP-based traffic – when a connection fails, HTTP clients automatically attempt a new connection, which the backup (now primary, active) server handles.

For on-premises deployments like our sample topology, NGINX Plus uses a high-availability solution based on keepalived and VRRP. The `nginx-ha-keepalived` package runs on both NGINX Plus instances. On both instances, you use the configuration wizard on each instance to specify the peer's IP address and the floating virtual IP address (VIP) they share, which is advertised to clients. You assign different priorities to the instances to indicate which one is normally the primary, active instance. If the active instance fails, the passive instance takes over the VIP and actively listens on it.

High availability on Cisco ACE requires configuring multiple VLANs. This is not required with NGINX Plus.

On the passive server the NGINX Plus configuration must be manually copied over with IP addresses changed.

Active/passive redundancy is exclusive to NGINX Plus.

Cisco ACE

```
interface vlan 1000
    description Management connectivity on VLAN 1000 and query
    interface VLAN
    ip address 172.25.91.110 255.255.255.0
    peer ip address 172.25.91.111 255.255.255.0
    alias ip address 172.25.91.112 255.255.255.0
    service-policy input REMOTE_MGMT_ALLOW_POLICY
    no shutdown
ft interface vlan 60
    ip address 10.10.60.10 255.255.255.0
    peer ip address 10.10.60.11 255.255.255.0
ft peer 1
    heartbeat interval 300
    heartbeat count 20
ft-interface vlan 60

ft group 1
    peer 1
    associate-context VC_WEB
    inservice
```

NGINX Plus

Begin by installing the `nginx-ha-keepalived` package:

On Ubuntu Server:

```
$ sudo apt-get install nginx-ha-keepalived
```

On CentOS/RHEL:

```
$ sudo yum install nginx-ha-keepalived
```

Then run the set-up script:

```
$ nginx-ha-setup
<snip>
Step 1: configuring internal management IP addresses.

In order to communicate with each other, both nodes must have at
least one IP address.

The guessed primary IP of this node is: 10.10.40.10

Do you want to use this address for internal cluster
communication? (y/n)
IP address of this host is set to: 10.10.40.10
Primary network interface: ens33

Now please enter IP address of a second node: 10.10.40.11

<snip>

Enter cluster IP address: 10.10.40.12

Please choose what the current node role is:
1) MASTER
2) BACKUP

(on the second node you should choose the opposite variant)

Press 1 or 2.
```

4 Performance and Security Optimizations

"Without exception, all of my biggest mistakes occurred because I moved too slowly."

–John Chambers, former Cisco CEO

NGINX offers many performance and security optimizations, most of which are not available in Cisco ACE. In this chapter we show how configure some of the optimizations after you finish migrating from Cisco ACE.

Redirecting All Traffic to HTTPS

Google and other security-focused entities recommend using SSL for all Internet-facing site traffic. This prevents a man-in-the-middle from intercepting traffic and possibly monitoring users or swapping in its own content instead of your pages. Google puts weight behind its recommendation by rewarding sites that encrypt all traffic with higher search rankings. Beginning in July 2018, Google Chrome version 68 will mark all HTTP sites as "not secure".

The first `server` block listens on port 80 forces clients to use HTTPS, by returning status code `301 (Moved Permanently)` to redirect clients that have sent requests on port 80 to the same URL but with **https** as the scheme instead of **http**. The second virtual server listens on port 443 for the SSL-encrypted traffic:

```
server {
    listen 80 default_server;
    server_name www.example.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl default_server;
    server_name www.example.com;
    ssl_certificate cert.crt;
    ssl_certificate_key cert.key;

    location / {
        proxy_set_header Host $host;
        proxy_pass http://SF_WEB;
    }
}
```

Enabling HTTP/2

HTTP/2 is the latest version of the HTTP protocol, standardized in 2015. Because it can pipeline requests and uses smaller HTTP headers, it performs better than HTTP/1.*x* in many situations. NGINX can act as an "HTTP/2 gateway" translating HTTP/2 back to HTTP/1.1 when communicating with real servers that do not support HTTP/2. HTTP/2 requires SSL.

To enable HTTP/2 support, add **`http2`** to the listen directive of an SSL-enabled virtual server.

```
server {
    listen 443 ssl http2 default_server;
    server_name www.example.com;
    ssl_certificate cert.crt;
    ssl_certificate_key cert.key;
}
```

Enabling Content Caching

Some of the world's largest CDNs, including MaxCDN, Fastly, and Instart Logic, take advantage of NGINX content caching. When Netflix decided to build its own CDN, it used NGINX open source at its core. In fact, it's estimated that nearly a third of North America's internet traffic flows through NGINX caches due to Netflix's popular streaming service. You can use NGINX caching to improve performance for both dynamic and static content.

```
proxy_cache_path /tmp/cache levels=1:2
        keys_zone=my_cache:10m max_size=10g
        inactive=60m use_temp_path=off;
server {
    location / {
        proxy_cache my_cache;
        proxy_set_header Host $host;
    proxy_pass http://SF_WEB;
    }
}
```

The **proxy_cache_path** directive sets the path and configuration of the cache, and the **proxy_cache** directive activates it:

- **proxy_cache_path** – Defines a 10 GB cache stored in **/tmp/cache**. Inactive elements in the cache are deleted after 60 minutes.
- **proxy_cache** – Enables caching for the location block it is in. In the example, it applies to all content on the site.

For more details on NGINX content caching, see A Guide to Caching with NGINX and NGINX Plus on our blog.

Enabling Keepalive Connections to the Server Farm

For maximum compatibility with applications, by default NGINX makes HTTP/1.0 connections to the real servers in the server farm. If the real servers support HTTP/1.1 (most do), we recommend instructing NGINX to use it. HTTP/1.1 supports keepalive TCP connections, and reusing connections for multiple requests improves performance by reducing the overhead of creating new TCP connections.

```
upstream SF_WEB {
    server 10.10.50.10:80; # content server web-one
    server 10.10.50.11:80; # content server web-two
    keepalive 32;
}
server {
    location / {
        proxy_http_version 1.1;
        proxy_set_header Connection “”;
        proxy_set_header Host $host;
        proxy_pass http://my_upstream;
    }
}
```

Explanation of the key directives:

- **keepalive** – Enables the TCP connection cache and sets the maximum number of TCP connections to keep open (here, 32).
- **proxy_http_version** – Upgrades connections to the server farm to HTTP/1.1.
- **proxy_set_header** – Enables keepalive TCP connections by clearing the Connection: Close HTTP header. By default NGINX uses HTTP/1.0 with `Connection: Close`.

5 MidwayUSA Case Study

"We had been pretty happy with our former [Cisco ACE] load balancer, but we were definitely ready to move. We were starting to get to the point where automation was becoming a struggle for us."

–Drew Turner, Web Operations Manager at MidwayUSA

Situation

Founded in 1977, MidwayUSA is a successful ecommerce retailer that sells Just About Everything® for shooting, hunting, and the outdoors. With more than 100,000 products and over 1.2 million active customers, MidwayUSA is a global leader in its market.

MidwayUSA has a strong focus company-wide on providing excellence for its customers, including a feature-rich, user-friendly, and fast website experience. It's the web operations team's responsibility to deliver the website with excellence. Until about four years ago, MidwayUSA used Cisco ACE for load balancing its web traffic, but the ACE product line was reaching end of life. At the same time, the team at MidwayUSA realized that to provide the best ecommerce experience for their customers, their load balancer needed more flexible configuration and automation capabilities.

"We had been pretty happy with our former load balancer, but we were definitely ready to move. We were starting to get to the point where automation was becoming a struggle for us. We also wanted more flexibility, including consistency across all our environments, including test. With our old hardware load balancer, we didn't have a test device that we could use. Instead we had to create special logic, which meant our test environment wasn't the same as production. We needed more flexibility and easier automation, so we began looking for a replacement load balancing solution," explains Drew Turner, web operations manager at MidwayUSA.

Solution

MidwayUSA's web operations team evaluated traditional hardware load balancers like F5 as well as software load balancers like HAProxy and NGINX Plus. MidwayUSA ultimately chose NGINX Plus for its combination of price, performance, features, and technical support.

"We looked at traditional vendors, as well as other solutions that peers in our industry are using, and NGINX Plus kept coming up. When we evaluated all the feature sets from our top three or four load balancing choices, we found that NGINX Plus outweighed everyone else. NGINX Plus has SSL offloading built in, great support from engineers who can add in the extra features we need and help us at 10 PM if we're having a problem, and all of the feature sets we were looking for. On top of that, it's very easy to get started with NGINX Plus, and the investment is significantly lower than hardware load balancers," notes Turner.

The process of migrating from Cisco ACE to NGINX Plus was smooth. "Moving to NGINX Plus was easy. We stood up our NGINX Plus instances, configured them, then we modified where the firewall pointed to for the NAT to do its conversion. There was some build time, lots of testing; and then the conversion process was just changing where the network address pointed to," notes Turner.

Overview
MidwayUSA is a privately held American online retailer of hunting and outdoor-related products.

Challenge
Cisco ACE hardware appliance going end-of-life. Manual processes taking up too much time.

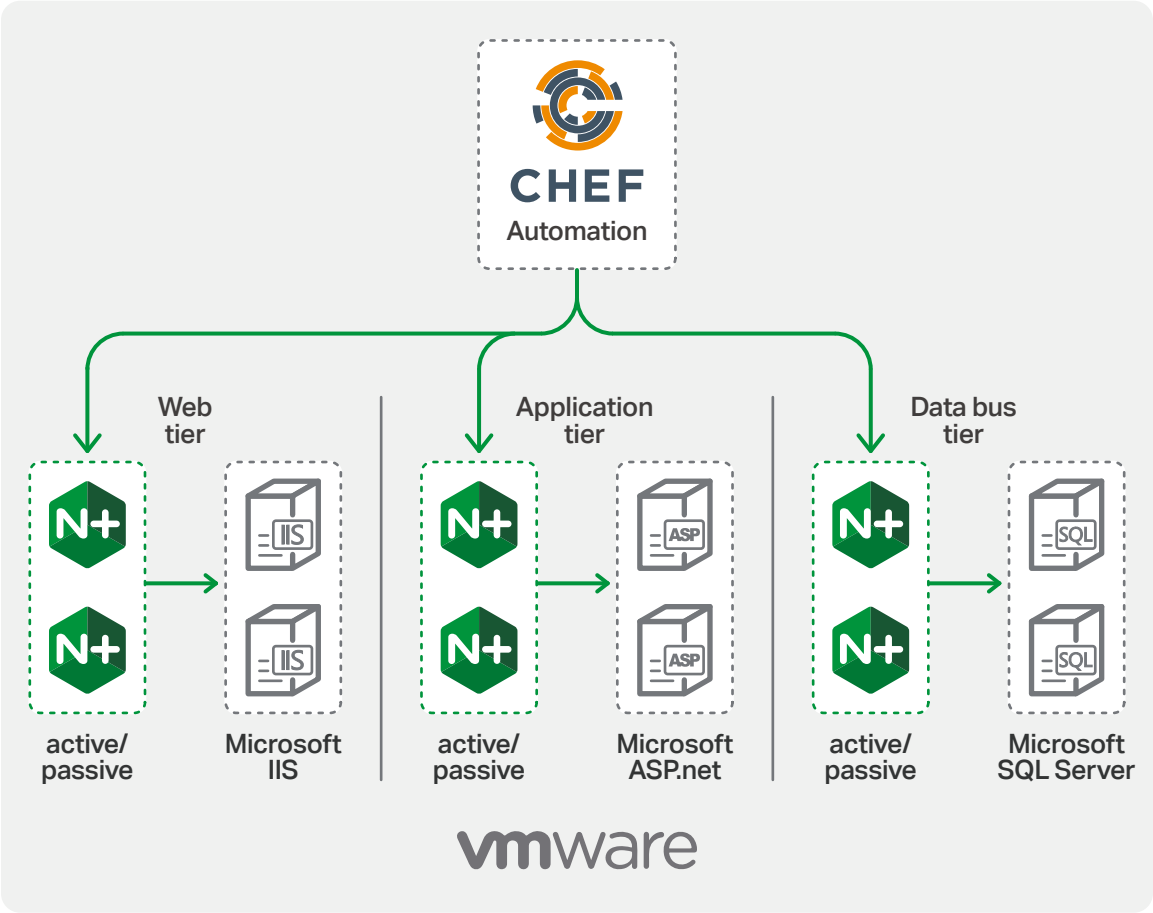
Solution
NGINX Plus with NGINX Chef cookbook gives MidwayUSA push button deployments to production.

Founded
1977

Headquarters
Columbia, Missouri

Implementation

MidwayUSA uses NGINX Plus as a load balancer throughout its multitiered architecture, including the web, application, and database layers. Each tier includes separate instances of NGINX Plus in a paired active-passive configuration for high availability, resulting in a total of six instances. Another three instances are used for development.



MidWayUSA's 3-tier architecture

At the entry point to its application, MidwayUSA uses NGINX Plus as a frontend load balancer to handle incoming web traffic. NGINX Plus load balances traffic to the web server tier, which currently uses Microsoft IIS. At the application tier (service layer), MidwayUSA uses another active-passive pair of NGINX Plus instances in almost the same configuration to route traffic to the backend ecommerce application which serves functions such as checkout, product pages, and search, and is written in C# using ASP.NET. At the database tier, NGINX Plus load balances the Microsoft SQL databases for the website. Having separate instances of NGINX Plus throughout the architecture makes it easy for the web operations team at MidwayUSA to achieve its goals.

"I'm a really big fan of being very modular. We could do a lot of this stuff with just one instance of NGINX Plus. But I like to have NGINX Plus deployed in separate demilitarized zones, with separate virtualized instances on each layer. It makes it easier when I want to spin up or down separate layers because I don't have to worry about moving everything at once," explains Turner.

Results

Substantially Less Time Spent on Site Maintenance

MidwayUSA's web operations team enjoys the clean, modular design of NGINX Plus, which makes it much easier to maintain and work with on a daily basis compared to its former hardware load balancers, and improves productivity as well.

"NGINX Plus is super easy to configure, which is very important for us. It's extremely modular, so we can share components across various sites. For instance, we share our **proxy.conf** configuration file over three of our frontend sites. It's really nice to make one change and have it work for all three sites," says Turner. "From a configuration standpoint, there's certainly been a significant reduction in our maintenance activities."

Instead of having one giant configuration file, NGINX Plus supports modular configuration files separated by function, improving readability and maintainability. The simplified configuration management, along with automation capabilities, has made the lives of the operations team much easier. Turner notes, "We're spending substantially less time on website management than with Cisco ACE. With NGINX Plus, we're able to spin things up and test them ridiculously fast."

Increasing Agility through Automation

"The web operations team likes to build fast. Using NGINX Plus and Chef, we're able to get machines up much, much faster than with hardware appliances," says Turner. "And because NGINX Plus is software, we're able to do a lot with Infrastructure as Code that we couldn't do before."

NGINX Plus has enabled MidwayUSA to increase infrastructure automation, which has shortened development cycles and improved agility. Using NGINX Plus with Chef automation, deploying changes to applications is as simple as clicking a button. Rather than deploying changes to production servers, MidwayUSA quickly spins up new NGINX Plus instances on staging servers using Chef. The staging deployment is then tested. If everything checks out, the production environment is changed to point to the staging servers, and NGINX Plus is reloaded. The result is a quick, "hitless" transition with minimal risk and no downtime.

MidWayUSA's website is powered by NGINX Plus

Better Response Times with Caching

"The caching capabilities of NGINX Plus are beyond anything we used before. With NGINX Plus there's the ability to cache anything and everything in super high detail," says Turner.

The incredibly granular level of caching made possible by NGINX Plus has increased website performance and responsiveness. Turner notes: "We've seen a really nice improvement in site performance from caching requests between our web and application layer. Cached responses are around 10 milliseconds, whereas uncached requests that go to database take on average 100 milliseconds, so we're able to shave a ton of time off those requests."

Besides caching internal application requests, MidwayUSA also uses NGINX Plus to cache results from its search bar, which is powered by a third-party provider. Cached search results are returned in 10 to 15 milliseconds, whereas queries that go to the search provider can take up to 700 milliseconds – a dramatic difference.

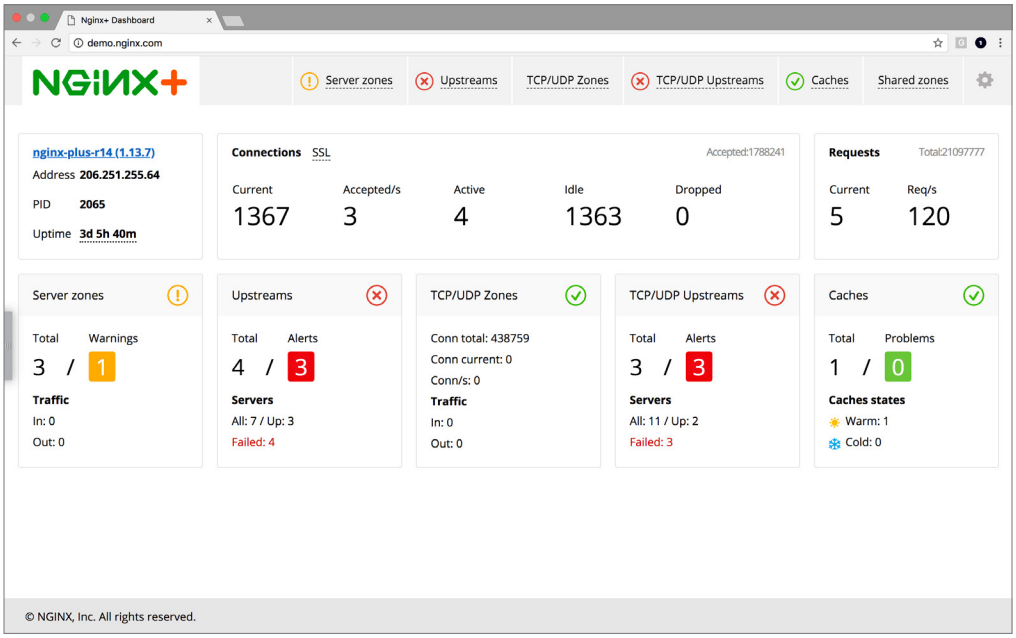
Perhaps most remarkably, MidwayUSA has found that its caching performance gains come at a minimal impact on CPU.

Turner explains, "We benchmarked NGINX Plus against Squid, a dedicated caching proxy. For the same workload, Squid would use 30% CPU whereas NGINX Plus used just 4%."

Superior Logging and Monitoring

With the extensive logging capabilities of NGINX Plus, MidwayUSA can better monitor its operations, both with complementary third-party software and with the NGINX Plus live activity monitoring dashboard.

"Because the logging is so capable in NGINX Plus, we can look at in-depth statistics such as response times, cache request rates, and cache hit ratios," explains Turner. "We do a lot of our monitoring with Splunk and we basically grab the logs from NGINX Plus to do that. With NGINX Plus we can get a lot more information a lot easier than we could with other solutions. We're also big fans of the NGINX Plus dashboard, which has incorporated suggestions from our team."



The NGINX Plus dashboard provides critical insights into application performance

Remarkably Easy A/B testing

NGINX Plus' sophisticated traffic routing features have enabled MidwayUSA to easily conduct A/B testing to see the effects of changes to the website.

After easily spinning up and deploying a test machine with the change to be tested, MidwayUSA can direct a percentage of traffic to the new version of the website by changing the load-balancing parameters of NGINX Plus.

"When you make a large change to the website, such as a new checkout process, if you deliver that to everyone, you impact 100% of customers. With A/B testing using NGINX Plus, we can set it so that a small percentage of customers go through the new process. Then we can evaluate metrics that are important to us and decide whether or not to fully deploy the change. For example, if we find that the conversion rate is significantly lower versus the control group, we've only impacted a small percentage of our customers and can easily roll things back. A/B testing with NGINX Plus helps us mitigate risks and improve our service level."

A/B testing has been critical for deploying large-scale changes, but because of the ease of which it can now be done, MidwayUSA is able to test even smaller changes, allowing it to fine-tune the user experience.

"We used to do our A/B testing through Sitespect, and that helped us test some of our large-scale changes, but it's so much faster and easier to conduct A/B testing with NGINX Plus that we're even able to test some small things like different checkout buttons or colors. We can test anything easily at this point," says Turner.

About MidwayUSA

Both country kids from Missouri, Larry and Brenda Potterfield turned their passion for shooting sports into a career by opening a small gun shop in 1977 that would eventually become MidwayUSA. They instilled family values like honesty, integrity, and respect for others into the business, and strive to maintain this culture with each employee added to their growing team. For over 39 years, MidwayUSA has maintained an unyielding focus on customer satisfaction and continues to offer Just About Everything for shooting, hunting, and the outdoors. For more information, visit www.midwayusa.com.

Appendix A:

Comparing Cisco ACE, NGINX, and NGINX Plus

Feature	Cisco ACE	NGINX	NGINX Plus
HTTP/TCP/UDP load balancer	✓	✓	✓
Round Robin, Hash, and Least Connections predictors	✓	✓	✓
Least time predictor	✓		✓
Active health probes	Match HTTP status only		Match HTTP status and resonse body
HTTP cookie stickiness	✓		✓
DNS service discovery integration			✓
Firewall load balancing	✓		
HTTP/2 gateway		✓	✓
SSL/TLS termination	✓	✓	✓
Dual-stack RSA/ECC		✓	✓
Perfect forward secrecy		✓	✓
Content cache		✓	✓
Web server		✓	✓

(continues)

Feature	Cisco ACE	NGINX	NGINX Plus
Security controls		✓	✓
Rate limiting	Connection, bandwidth	Request, connection, bandwidth	Request, connection, bandwidth
HTTP Basic authentication		✓	✓
JSON Web Token (JWT) authentication			✓
OpenID Connect Single Sign on (SSO)			✓
ModSecurity WAF module		Compile from source	Prebuilt module
Kubernetes Ingress controller		✓	✓
Programmability	✓	✓	✓
Scripting	TCL	JavaScript, Lua	JavaScript, Lua
HTTP API			✓
High availability	Full connection mirroring		VRRP-based

Appendix B:

Document Revison History

Version	Date	Description
1.0	2018-03-15	Initial release