Load Balancing 101: Nuts and Bolts

Introduction

In today's dynamic, application-centric marketplace, organizations are under more and more pressure to deliver the information, services, and experiences their customers expect—and to do it quickly, reliably, and securely. Key network and application functions, such as load balancing, encryption, acceleration, and security, can be provided via Application Delivery Controllers (ADC), which are physical or virtual appliances functioning as proxies for physical servers. With the explosion of applications, as well as the demands placed on organizations by the rigorous cycle of Continuous Integration and Continuous Deployment (CI/CD), it's no wonder that the market for ADCs is projected to reach $2.9 billion a year by 2020.[1]

But before heading into the future, let's look at how we got here. Network-based load balancing is the essential foundation upon which ADCs operate. In the mid-1990s, the first load balancing hardware appliances began helping organizations scale their applications by distributing workloads across servers and networks. These first devices were application-neutral and resided outside of the application servers themselves, which means they could load balance using straightforward network techniques. In essence, these devices would present a "virtual IP address" to the outside world, and when users attempted to connect, they would forward the connection to the most appropriate real server doing bi-directional network address translation (NAT).

However, with the advent of virtualization and cloud computing, a new iteration of load balancing ADCs arrived as software-delivered virtual editions intended to run on hypervisors. Today, virtual appliances can deliver application

Chat with Codey

eliminate much of the complexity involved in moving application services between virtual, cloud, and hybrid environments, allowing organizations to quickly and easily spin up application services in private or public cloud environments.

The newest trend to hit the data center is containerization, which is a method of application virtualization that helps in deploying and running distributed applications. The process isolates applications and contains them in clear delineated memory spaces on a shared OS, which not only makes developing and deploying an application easier than building a virtual appliance, but also make it quicker. Due to the dramatic improvements in portability and performance, containerization could provide businesses with greater scalability and agility. In the future, container architectures could also help organizations take better advantage of different cloud environments.

Today's ADCs evolved from the first load balancers through the service virtualization process. And now, with software-only virtual editions, ADCs can not only improve availability, but also help organizations deliver the scalable, high-performance, and secure applications that their business requires. In the end, though, all these virtualized application services, shared infrastructure deployments, and intelligent routing capabilities wouldn't be possible without the solid foundation of load balancing technology.

To understand how enterprises can better address the complex challenges of the dynamically evolving marketplace, let's explore the foundation of application delivery: Load Balancing 101.

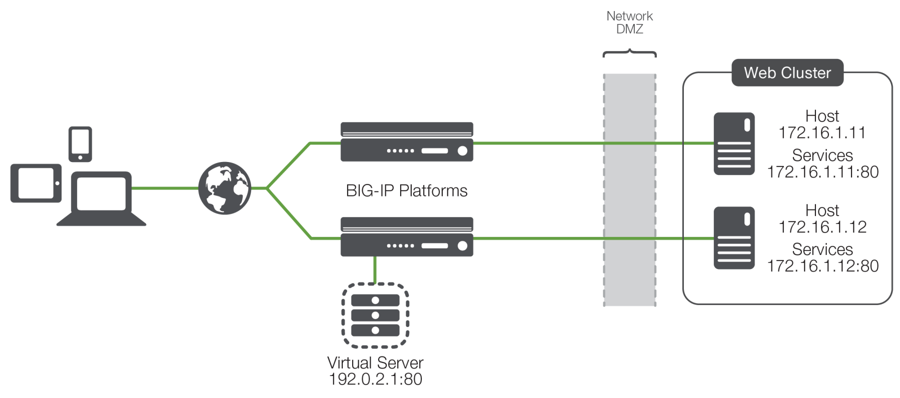

Figure 1: Network-based load balancing appliances.

The basics: terminology

Before we get started, let's review the basic terminology of load balancing. This would be easier if everyone used the same lexicon; unfortunately, every vendor of load balancing devices (and, in turn, ADCs) seems to use different terminology. With a little explanation, however, we can cut through the confusion.

Chat with Codey

Node, host, member, and server

host, member, or server; some have all four, but they mean different things. There are two basic concepts that these terms all try to express. One concept—usually called a node or server—is the idea of the physical or virtual server itself that will receive traffic from the ADC. This is synonymous with the IP address of the physical server and, in the absence of a load balancer, would be the IP address that the server name (for example, www.example.com) would resolve to. For the remainder of this paper, we will refer to this concept as the host.

The second concept is expressed by the term "member" (unfortunately also called a node by some manufacturers). A member is usually a little more defined than a host in that it includes the TCP port of the actual application that will be receiving traffic. For instance, a host named www.example.com may resolve to an address of 172.16.1.10, which represents the host, and may have an application (a web server) running on TCP port 80, making the member address 172.16.1.10:80. Simply put, the member includes the definition of the application port as well as the IP address of the physical/virtual server. For the remainder of this paper, we will refer to this as the service.

Why all the complexity? Because the distinction between a server and the application services running on it allows the load balancer to individually interact with the applications rather than the underlying hardware or hypervisor, in a datacenter or in the cloud. A host (172.16.1.10) may have more than one service available (HTTP, FTP, DNS, etc.). By defining each application uniquely (172.16.1.10:80, 172.16.1.10:21, and 172.16.1.10:53, for example), the ADC can apply unique load balancing and health monitoring (a concept we'll discuss later) based on the services instead of the host.

Remember, most load balancing—based technology uses one term to represent the host, or physical server, and another to represent the services available on it—in this case, simply host and services.

Pool, cluster, and farm

Load balancing allows organizations to distribute inbound application traffic across multiple back-end destinations, including deployments in public or private clouds. It is therefore a necessity to have the concept of a collection of back-end destinations. Clusters, as we will refer to them (they're also known as pools or farms), are collections of similar services available on any number of hosts. For instance, all the services that offer the company web page would be collected into a cluster called "company web page" and all the services that offer e-commerce services would be collected into a cluster called "e-commerce".

Chat with Codey

Virtual server

virtual, or container). Combined with a virtual IP address, this is the application endpoint that is presented to the outside world.

Putting it all together

With an understanding of these terms, we've got the basics of load balancing. The load balancing ADC presents virtual servers to the outside world. Each virtual server points to a cluster of services that reside on one or more physical hosts.

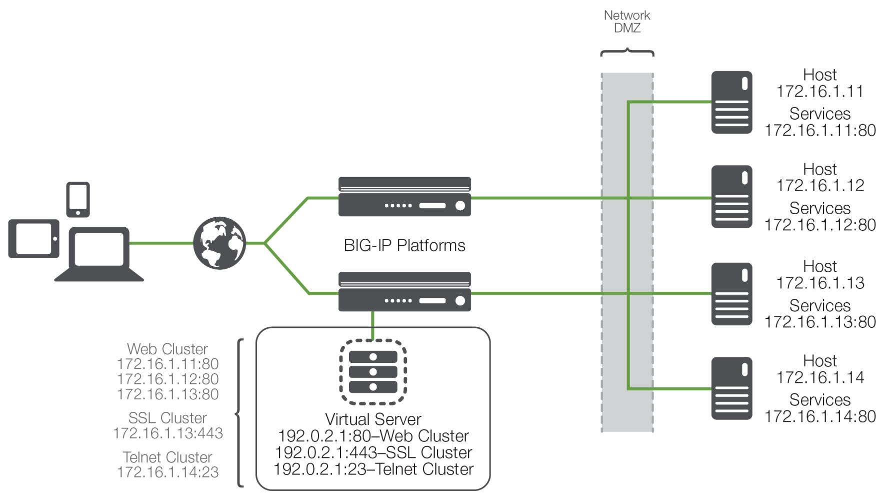

Figure 2: Application delivery comprises four basic concepts—virtual servers, clusters, services, and hosts.

While Figure 2 may not be representative of any real-world deployment, it does provide the elemental structure for continuing a discussion about the process of load balancing and application delivery.

How load balancing works

With this common vocabulary established, let's examine the simple transaction of how the application is delivered to the customer. As depicted, the load balancing ADC will typically sit in-line between the client and the hosts that provide the services the client wants to use. As with most things in application delivery, this positioning is not a rule, but more of a best practice in any kind of deployment. Let's also assume that the ADC is already configured with a virtual server that points to a cluster consisting of two service points. In this deployment scenario, it is common for the hosts to have a return route that points back to the load balancer so that return traffic will be processed through it on its way back to the client.

The basic application delivery transaction is as follows:

- The client attempts to connect with the service.
- The ADC accepts the connection, and after deciding which host should receive the connection, changes the destination IP (and possibly port) to match the service of the selected host (note that the source IP of the client is not touched).
- The host accepts the connection and responds back to the original source, the client, via its default route, the

Chat with Codey

now changes the source IP (and possible port) to match the virtual server IP and port, and forwards the packet back to the client.
- The client receives the return packet, believing that it came from the virtual server, and continues the process.

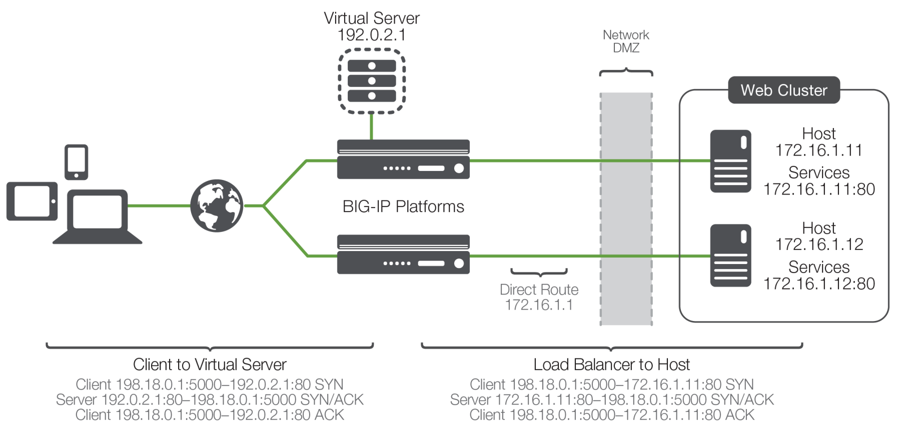
Virtual Server
192.0.2.1
Network
DMZ
Web Cluster
Host
172.16.1.11
Services
172.16.1.11:80
BIG-IP Platforms
Host
172.16.1.12
Services
172.16.1.12:80
Direct Route
172.16.1.1
Client to Virtual Server
Client 198.18.0.1:5000–192.0.2.1:80 SYN
Server 192.0.2.1:80–198.18.0.1:5000 SYN/ACK
Client 198.18.0.1:5000–192.0.2.1:80 ACK
Load Balancer to Host
Client 198.18.0.1:5000–172.16.1.11:80 SYN
Server 172.16.1.11:80–198.18.0.1:5000 SYN/ACK
Client 198.18.0.1:5000–172.16.1.11:80 ACK

Figure 3: A basic load balancing transaction.

This very simple example is relatively straightforward, but there are a couple of key elements to note. First, as far as the client knows, it sends packets to the virtual server and the virtual server responds—simple. Second, the NAT takes place. This is where the ADC replaces the destination IP sent by the client (of the virtual server) with the destination IP of the host to which it has chosen to load balance the request. Third is the part of this process that makes the NAT "bi-directional". The source IP of the return packet from the host will be the IP of the host; if this address were not changed and the packet was simply forwarded to the client, the client would be receiving a packet from someone it didn't request one from, and would simply drop it. Instead, the load balancer, remembering the connection, rewrites the packet so that the source IP is that of the virtual server, thus solving this problem.

The application delivery decision

Usually at this point, two questions arise: How does the load balancing ADC decide which host to send the connection to? And what happens if the selected host isn't working?

Let's discuss the second question first. What happens if the selected host isn't working? The simple answer is that it doesn't respond to the client request and the connection attempt eventually times out and fails. This is obviously not a preferred circumstance, as it doesn't ensure high availability. That's why most load balancing technology includes some level of health monitoring to determine whether a host is actually available before attempting to send connections to it.

There are multiple levels of health monitoring, each with increasing granularity and focus. A basic monitor would simply ping the host itself. If the host does not respond to the ping, it is a good assumption that any services defined on the host are probably down and should be removed from the

Chat with Codey

itself is working. Therefore, most devices can do "service pings" of some kind, ranging from simple TCP connections all the way to interacting with the application via a scripted or intelligent interaction. These higher-level health monitors not only provide greater confidence in the availability of the actual services (as opposed to the host), but they also allow the load balancer to differentiate between multiple services on a single host. The load balancer understands that while one service might be unavailable, other services on the same host might be working just fine and should still be considered as valid destinations for user traffic.

This brings us back to the first question: How does the ADC decide which host to send a connection request to? Each virtual server has a specific dedicated cluster of services (listing the hosts that offer that service) that makes up the list of possibilities. Additionally, the health monitoring modifies that list to make a list of "currently available" hosts that provide the indicated service. It is this modified list from which the ADC chooses the host that will receive a new connection. Deciding on the exact host depends on the load balancing algorithm associated with that particular cluster. Some of these algorithms include least connections, dynamic ratio and a simple round robin where the load balancer simply goes down the list starting at the top and allocates each new connection to the next host; when it reaches the bottom of the list, it simply starts again at the top. While this is simple and very predictable, it assumes that all connections will have a similar load and duration on the back-end host, which is not always true. More advanced algorithms use things like current-connection counts, host utilization, and even real-world response times for existing traffic to the host in order to pick the most appropriate host from the available cluster services.

Sufficiently advanced application delivery systems will also be able to synthesize health monitoring information with load balancing algorithms to include an understanding of service dependency. This is mainly useful in cases when a single host has multiple services, all of which are necessary to complete the user's request. In such an instance you don't want a user going to a host that has one service operational but not the other. In other words, if one service fails on the host, you also want the host's other service to be taken out of the cluster list of available services. This functionality is increasingly important as services become more differentiated with HTML and scripting.

To load balance or not to load balance?

The portion of load balancing that involves picking an available service when a client initiates a transaction request is only half of the solution. Once the connection is established, the ADC must keep track of whether the following traffic from that user should be load balanced.

Chat with Codey

maintenance and persistence.

Connection maintenance

If the user is trying to utilize a long-lived TCP connection (Port 21: FTP, Port 23: Telnet, or other) that doesn't immediately close, the load balancer must ensure that multiple data packets carried across that connection do not get load balanced to other available service hosts. This is connection maintenance and requires two key capabilities. The first is the ability to keep track of open connections and the host service they belong to. Second, the load balancer must be able to continue to monitor that connection so the connection table can be updated when the connection closes. This is rather standard fare for most ADCs.

Persistence

Increasingly more common, however, is when the client uses multiple short-lived TCP connections (for example, Port 80: HTTP) to accomplish a single task. In some cases, like standard web browsing, it doesn't matter and each new request can go to any of the back-end service hosts; however, there are many instances (XML, e-commerce, and so on) where it is extremely important that multiple connections from the same user go to the same back-end service host and not be load balanced. This concept is called persistence, or server affinity.

There are multiple ways to address this, depending on the protocol and the desired results. For example, in modern HTTP transactions, the server can specify a "keep-alive" connection, which turns those multiple short-lived connections into a single long-lived connection, which can be handled just like the other long-lived connections. However, this provides only a little relief, mainly because, as the use of web and mobile services increases, keeping all the connections open longer than necessary strains the resources of the entire system. That's why today—for the sake of scalability and portability—many organizations are moving toward building stateless applications that rely on APIs. This basically means that the server will forget all session information to reduce the load on the resources and in these cases, the state is maintained by passing session IDs as well as through the concept of persistence.

One of the most basic forms of persistence is source-address affinity, which involves simply recording the source IP address of incoming requests and the service host they were load balanced to, and making all future transactions go to the same host. Two ways to accomplish this are by using SSL session IDs and cookies. SSL persistence tracks SSL session using SSL session IDs, which means that even when the client's IP address changes, the load balancer will recognize the session being persistent based on session ID. Cookie-based persistence offers the option of inserting a

Chat with Codey

connection goes to the correct server.

Today, the intelligence of ADCs allows organizations to open the data packets and create persistence tables for virtually anything within them. This enables them to use unique information, such as user name, to maintain persistence. However, organizations must ensure that this identifiable client information will be present in every request made, as any packets without it will not be persisted and will be load balanced again, most likely breaking the application.

Conclusion

In the beginning, load balancing focused on distributing workloads throughout the network and ensuring the availability of applications and services. As the technology evolved, however, load balancers became platforms for application delivery, ensuring that an organization's critical applications were highly available and secure. While basic load balancing remains the foundation of application delivery, modern ADCs offer much more enhanced functionality.

Enterprises realize that simply being able to reach an application doesn't make it usable—and unusable applications mean wasted time and money for the organization deploying them. That's where the modern ADC comes in, allowing organizations to consolidate network-based services like SSL/TLS offload, caching, compression, rate-shaping, intrusion detection, application firewalls, and even remote access into a single strategic point that can be shared and reused across all application services and all hosts to create a virtualized Application Delivery Network. This allows network, application, and operations teams better respond to business demands for shorter delivery timelines and greater scalability—while never sacrificing the need for security.

If you would like to learn more about how advanced application delivery works and the future of ADCs, read The Evolution of Application Delivery Controllers and Go Beyond Plain Old Load Balancing.

[1] http://www.strategyr.com/MarketResearch/Application_Delivery_Controllers_ADC_Market_Trends.asp

PUBLISHED MAY 10, 2017

CONNECT WITH F5

Chat with Codey

The latest in application threat intelligence.

Go to F5 Labs

The F5 community for discussion forums and expert articles.

Go to DevCentral

F5 NEWSROOM

News, F5 blogs, and more.

Go to the newsroom

Secure and Deliver Extraordinary Digital Experiences

F5's portfolio of automation, security, performance, and insight capabilities empowers our customers to create, secure, and operate adaptive applications that reduce costs, improve operations, and better protect users. Learn more ›

©2023 F5, Inc. All rights reserved.

Chat with Codey