

By: Erika Heidi

Subscribe

Share

Contents



What is High Availability?

Updated November 4, 2016 62k HIGH AVAILABILITY LOAD BALANCING CONCEPTUAL

27

Introduction

With an increased demand for reliable and performant infrastructures designed to serve critical systems, the terms scalability and high availability couldn't be more popular. While handling increased system load is a common concern, decreasing downtime and eliminating single points of failure are just as important. High availability is a quality of infrastructure design at scale that addresses these latter considerations.

In this guide, we will discuss what exactly high availability means and how it can improve your infrastructure's reliability.

What Is High Availability?

In computing, the term availability is used to describe the period of time when a service is available, as well as the time required by a system to respond to a request made by a user. High availability is a quality of a system or component that assures a high level of operational performance for a given period of time.

Measuring Availability

Availability is often expressed as a percentage indicating how much uptime is expected from a particular system or component in a given period of time, where a value of 100% would indicate that the system never fails. For instance, a system that guarantees 99% of availability in a period of one year can have up to 3.65 days of downtime (1%).

These values are calculated based on several factors, including both scheduled and unscheduled maintenance periods, as well as the time to recover from a possible system failure.

How Does High Availability Work ?

SCROLL TO TOP

High availability functions as a failure response mechanism for infrastructure. The way that it works is quite simple conceptually but typically requires some specialized software and configuration.

When Is High Availability Important ?

When setting up robust production systems, minimizing downtime and service interruptions is often a high priority. Regardless of how reliable your systems and software are, problems can occur that can bring down your applications or your servers.

Implementing high availability for your infrastructure is a useful strategy to reduce the impact of these types of events. Highly available systems can recover from server or component failure automatically.

What Makes a System Highly Available?

One of the goals of high availability is to eliminate single points of failure in your infrastructure. A single point of failure is a component of your technology stack that would cause a service interruption if it became unavailable. As such, any component that is a requisite for the proper functionality of your application that does not have redundancy is considered to be a single point of failure.

To eliminate single points of failure, each layer of your stack must be prepared for redundancy. For instance, imagine you have an infrastructure consisting of two identical, redundant web servers behind a load balancer. The traffic coming from clients will be equally distributed between the web servers, but if one of the servers goes down, the load balancer will redirect all traffic to the remaining online server.

The web server layer in this scenario is not a single point of failure because:

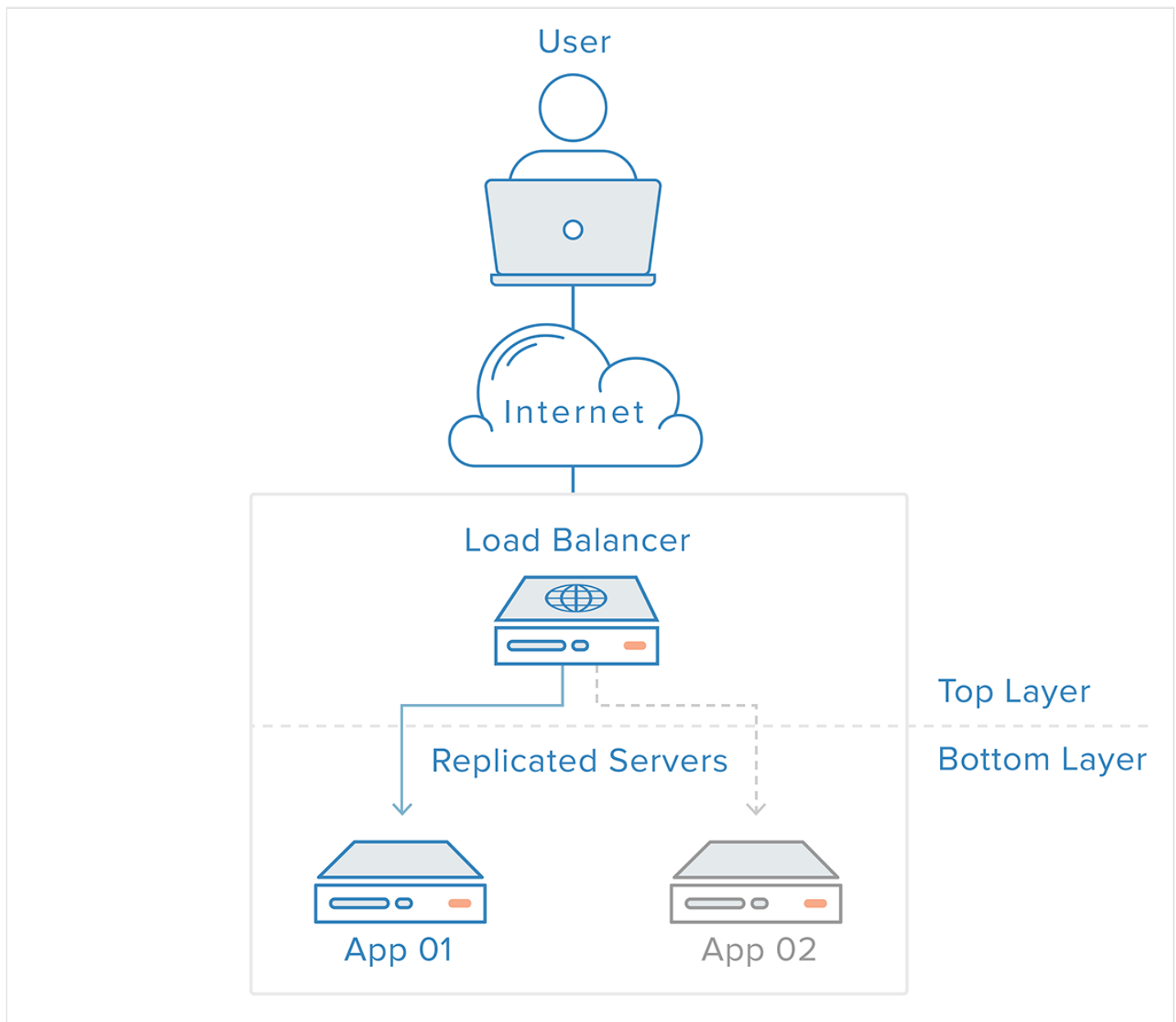
- redundant components for the same task are in place
- the mechanism on top of this layer (the load balancer) is able to detect failures in the components and adapt its behavior for a timely recovery

But what happens if the load balancer goes offline?

With the described scenario, which is not uncommon in real life, the load balancing layer itself remains a single point of failure. Eliminating this remaining single point of failure, however, can be challenging; even though you can easily configure an additional load balancer to achieve redundancy, there isn't an obvious point above the load balancers to implement failure detection and recovery.

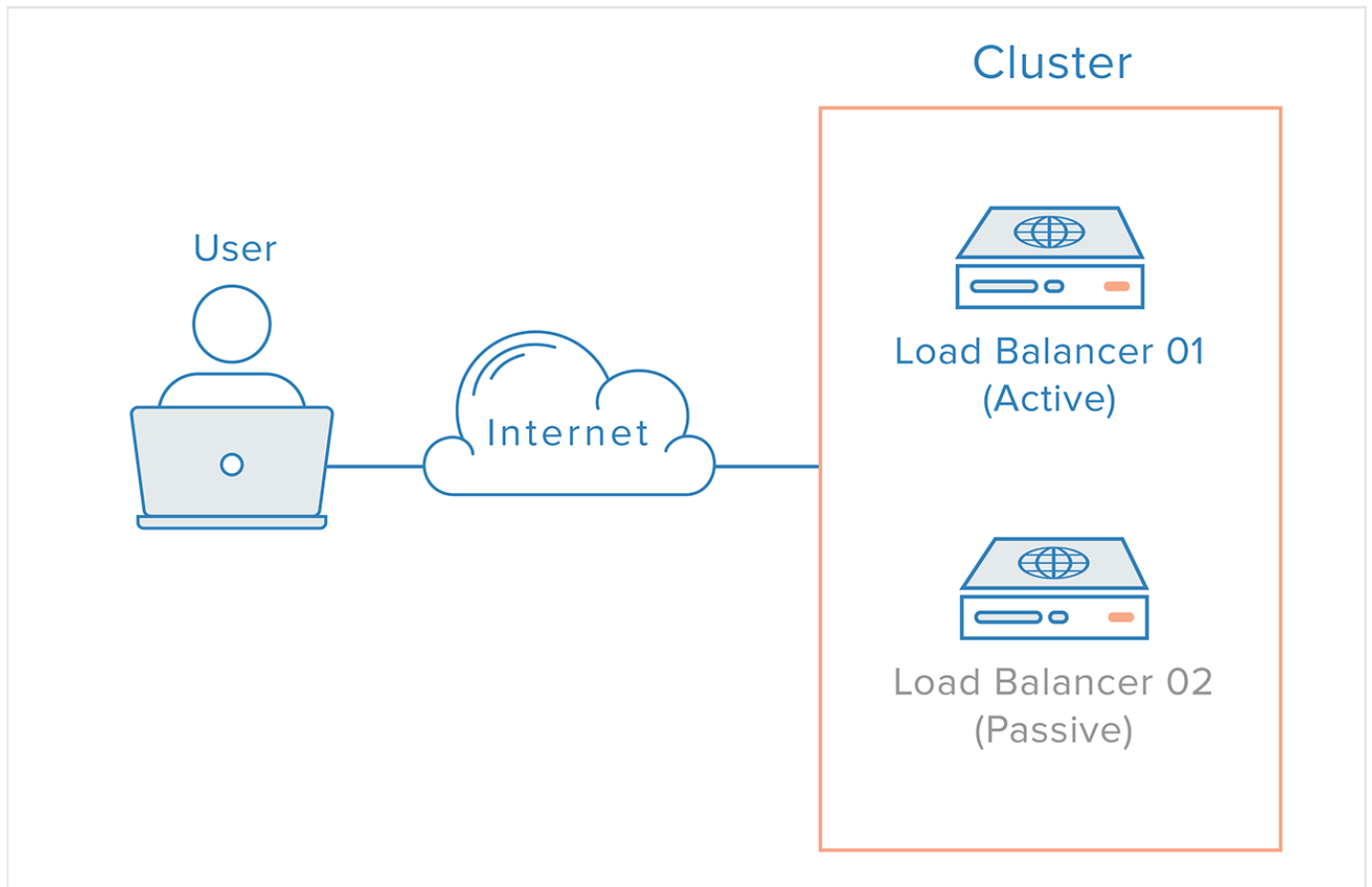
Redundancy alone cannot guarantee high availability. A mechanism must be in place for detecting failures and taking action when one of the components of your stack becomes unavailable.

Failure detection and recovery for redundant systems can be implemented using a top-to-bottom approach: the layer on top becomes responsible for monitoring the layer immediately beneath it for failures. In our previous example scenario, the load balancer is the top layer. If one of the web servers (bottom layer) becomes unavailable, the load balancer will stop redirecting requests for that specific server.



This approach tends to be simpler, but it has limitations: there will be a point in your infrastructure where a top layer is either nonexistent or out of reach, which is the case with the load balancer layer. Creating a failure detection service for the load balancer in an external server would simply create a new single point of failure.

With such a scenario, a distributed approach is necessary. Multiple redundant nodes must be connected together as a cluster where each node should be equally capable of failure detection and recovery.

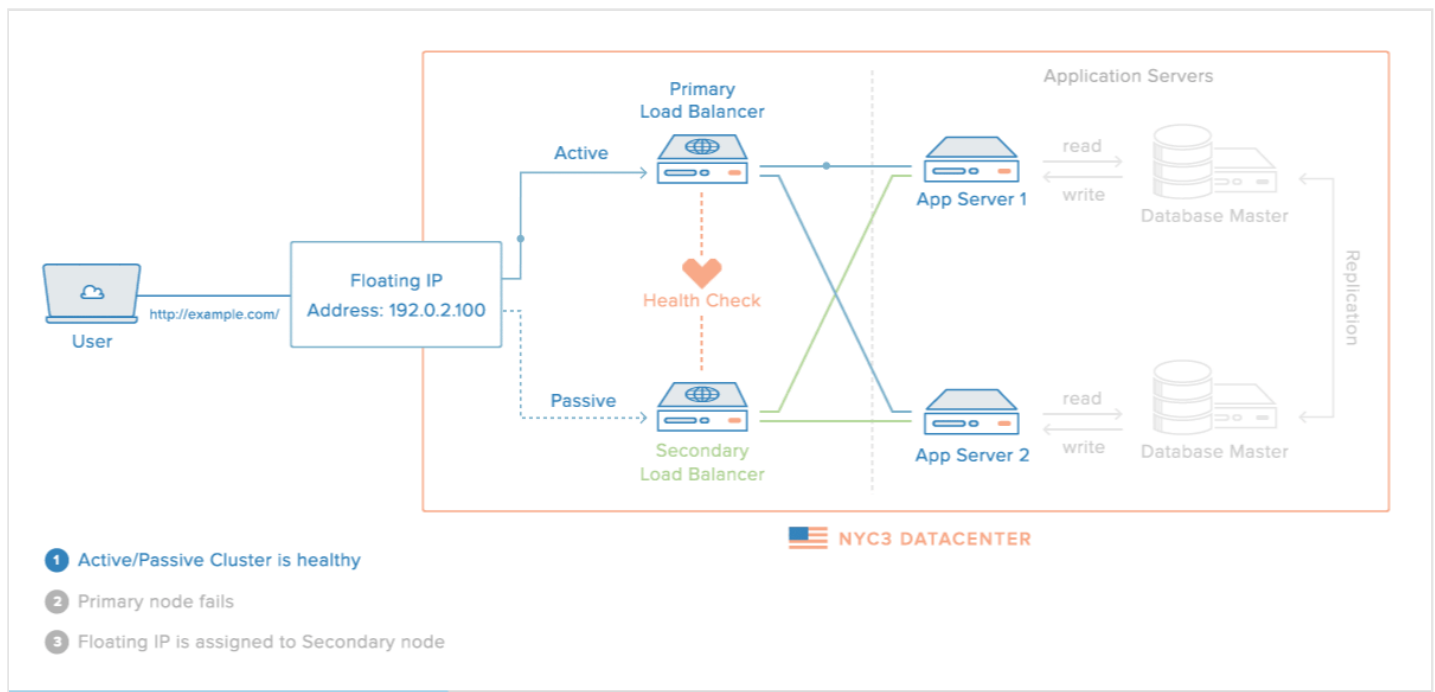


For the load balancer case, however, there's an additional complication, due to the way nameservers work. Recovering from a load balancer failure typically means a failover to a redundant load balancer, which implies that a DNS change must be made in order to point a domain name to the redundant load balancer's IP address. A change like this can take a considerable amount of time to be propagated on the Internet, which would cause a serious downtime to this system.

A possible solution is to use DNS round-robin load balancing. However, this approach is not reliable as it leaves failover to the client-side application.

A more robust and reliable solution is to use systems that allow for flexible IP address remapping, such as floating IPs. On demand IP address remapping eliminates the propagation and caching issues inherent in DNS changes by providing a static IP address that can be easily remapped when needed. The domain name can remain associated with the same IP address, while the IP address itself is moved between servers.

This is how a highly available infrastructure using Floating IPs looks like:



What System Components Are Required for High Availability?

There are several components that must be carefully taken into consideration for implementing high availability in practice. Much more than a software implementation, high availability depends on factors such as:

- **Environment:** if all your servers are located in the same geographical area, an environmental condition such as an earthquake or flooding could take your whole system down. Having redundant servers in different datacenters and geographical areas will increase reliability.
- **Hardware:** highly available servers should be resilient to power outages and hardware failures, including hard disks and network interfaces.
- **Software:** the whole software stack, including the operating system and the application itself, must be prepared for handling unexpected failure that could potentially require a system restart, for instance.
- **Data:** data loss and inconsistency can be caused by several factors, and it's not restricted to hard disk failures. Highly available systems must account for data safety in the event of a failure.
- **Network:** unplanned network outages represent another possible point of failure for highly available systems. It is important that a redundant network strategy is in place for possible failures.

What Software Can Be Used to Configure High Availability?

Each layer of a highly available system will have different needs in terms of software and configuration. However, at the application level, load balancers represent an essential piece of software for creating any high availability setup.

HAProxy (High Availability Proxy) is a common choice for load balancing, as it can handle load balancing at multiple layers, and for different kinds of servers, including database servers.

Moving up in the system stack, it is important to implement a reliable redundant solution for your application entry point, normally the load balancer. To remove this single point of failure, as mentioned before, we need to implement a cluster of load balancers behind a Floating IP. Corosync and Pacemaker are popular choices for creating such a setup, on both Ubuntu and CentOS servers.

Conclusion

High availability is an important subset of reliability engineering, focused towards assuring that a system or component has a high level of operational performance in a given period of time. At a first glance, its implementation might seem quite complex; however, it can bring tremendous benefits for systems that require increased reliability.

Get started on DigitalOcean with free \$10 credit

Cheers to you, voracious reader of Community Tutorials. You've read five tutorials in the last week! If you haven't already spun up servers on DO, here's \$10 to give it a try.

[CREATE AN ACCOUNT TO REDEEM CREDIT](#)

Related Tutorials

How To Create a High Availability Setup with Heartbeat and Floating IPs on Ubuntu 16.04

3 Strategies for Minimizing Downtime

How to Install and Configure Zabbix to Securely Monitor Remote Servers on Ubuntu 16.04

How To Configure SSL Passthrough on DigitalOcean Load Balancers

What is Load Balancing?

10 Comments

Leave a comment...


[Log In to Comment](#)

 [shutch190](#) MOD March 20, 2016

1 First line is a massive turn off for readers. So uninviting, basically says "There is a long, boring and technically detailed article filled with jargon coming now"

 [zaltsman](#) MOD March 22, 2016

0 [@shutch190](#) Thank you for reader feedback - that wasn't what we were going for. Which part came off uninviting, interested in knowing so we can improve?

 [shutch190](#) March 24, 2016

1 The first line. The introduction is actually not needed. The title of the article has brought me to the page - what is high availability? I want this information, not a long winded intro.

 [shutch190](#) March 24, 2016

[SCROLL TO TOP](#)

° Maybe it is a SEO tactic...having the keyword as the title and in a h1 and h2 tags and saturating that keyword...

^ [Brun](#) March 20, 2016

2 Very helpful. Thanks.

Erica, what program You use to drawing infrastructure?

^ [lamek](#) February 3, 2017

0 +1. Very interested in the tool you used to create your images and the animated gif.

^ [4lvin](#) April 25, 2016

0 Thanks for the tutorial Erika.

May we know a bit more about the actual implementation of it please? As in another tutorial, may be? :) Or at least please let me know a few points here:

1. According to the diagram (and from the D.O point of view), would the two "Load-balancers" be two Droplets (VMs)? And then use Nginx? Or use HAProxy?
2. Should we consider implementing the "Floating IP" setup by ourselves (using which tools?) or use the Digital Ocean's Floating IP new feature available right now, so that we don't need to worry about that part?

I'd love to learn more on this :)

Thanks with regards.

^ [MosziNet](#) May 31, 2016

- 1 So what piece of monitoring software should we use based on which we can implement the ip address remapping?
2. Will IP remapping work across geography zones too? If not, how to solve this?

^ [bidstarboii](#) November 4, 2016

0 Thanks for the article. There's a grammatical error in this sentence:

"A possible solution is to use DNS round-robin load balancing. However, this approach is not reliable as it leaves failover the the client-side application."

Just search for "the the".

^ [NicksonYap](#) February 17, 2017

1 Is it true floating IP does not work across datacenters?
Say, one server in New York and another in Singapore?

About DNS round-robin, the article mentions "it leaves failover to the client-side application."

Does this mean that if the browser does not look for other IP address listed in the record, then it's taken as server down?



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

SCROLL TO TOP