DevNet Associate v1.0

Networks for Application Development and Security

6.4.1

Introduction

These days, you must take networking into account for all but the simplest of use cases. This is especially true when it comes to cloud and container deployments. Here are some of the applications you need to consider when it comes to cloud deployment:

- Firewalls
- Load balancers
- DNS
- Reverse proxies

6.4.2

Firewall

Firewalls are a computer's most basic defense against unauthorized access by individuals or applications. They can take any number of forms, from a dedicated hardware device to a setting within an individual computer's operating system.

At its most basic level, a firewall accepts or rejects packets based on the IP addresses and ports to which they're addressed. For example, consider a web server. This server has on it the actual web server software, as well as the application that represents the site and the database that contains the content the site displays. Without a firewall, the server could be accessed in multiple ways:

Different applications access a server in different ways

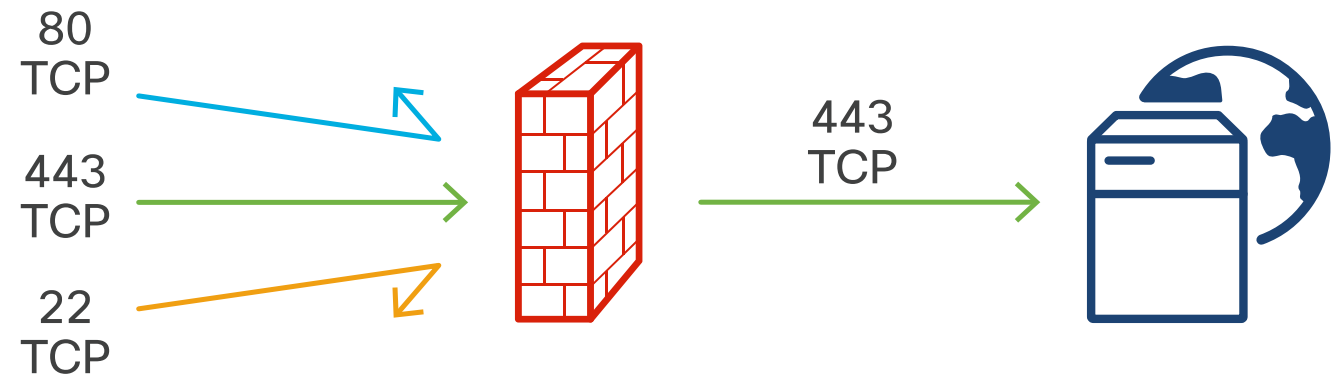
- A web browser can access the web application with an HTTP request to port 80 or an HTTPS request to port 443
- A database client can access the database with a UDP request to port 5000
- An SSH client can log into the server itself with a TCP request to port 22

But is that really what you want? You definitely want to access the web application, though perhaps you only want HTTPS requests. You definitely do NOT want anyone to access the database directly; in this case, only the web application really needs that access. You might want to be able to log in to the server using an SSH client, rather than having to have physical access to the machine.

To accomplish this, set up a firewall with specific “rules”, which are layered on top of each other. So for example, you may have this rule:

- Deny all access to anyone, with these rules...
  - Allow TCP requests to port 443 from anyone
  - Block all TCP requests to port 22
  - Block all TCP requests to port 80

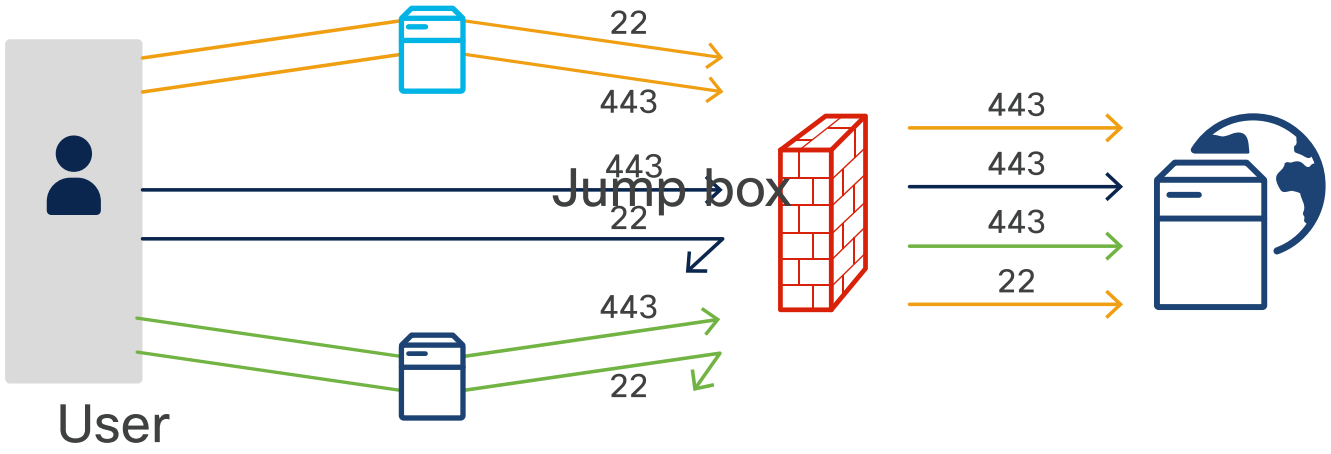
(Both HTTPS and SSH use TCP requests.)



A firewall can allow some connections and reject others.

So in this case, the only access to the machine would be those HTTPS requests. Anything else would be blocked by the firewall.

In some cases, you do want to enable access, but not from just anybody. For example, you might set up your systems so that logins to sensitive systems can only come from a single machine. This is called a "jump box", and everyone must log into that server first, then log into the target machine from there. A jump box can be used to provide additional access while still providing an additional layer of security.



A jump box and firewall combination.

For example, if your jump box had an internal IP address of 172.0.30.42, your firewall rules might look like this:

- Deny all access to anyone, except...
  - Allow TCP requests to port 443 from anyone
  - Allow TCP requests to port 22 from 172.0.30.42 only
  - Allow UDP requests to port 5000 from 172.0.30.42 only

With software deployments, here are some things to consider when it comes to firewalls:

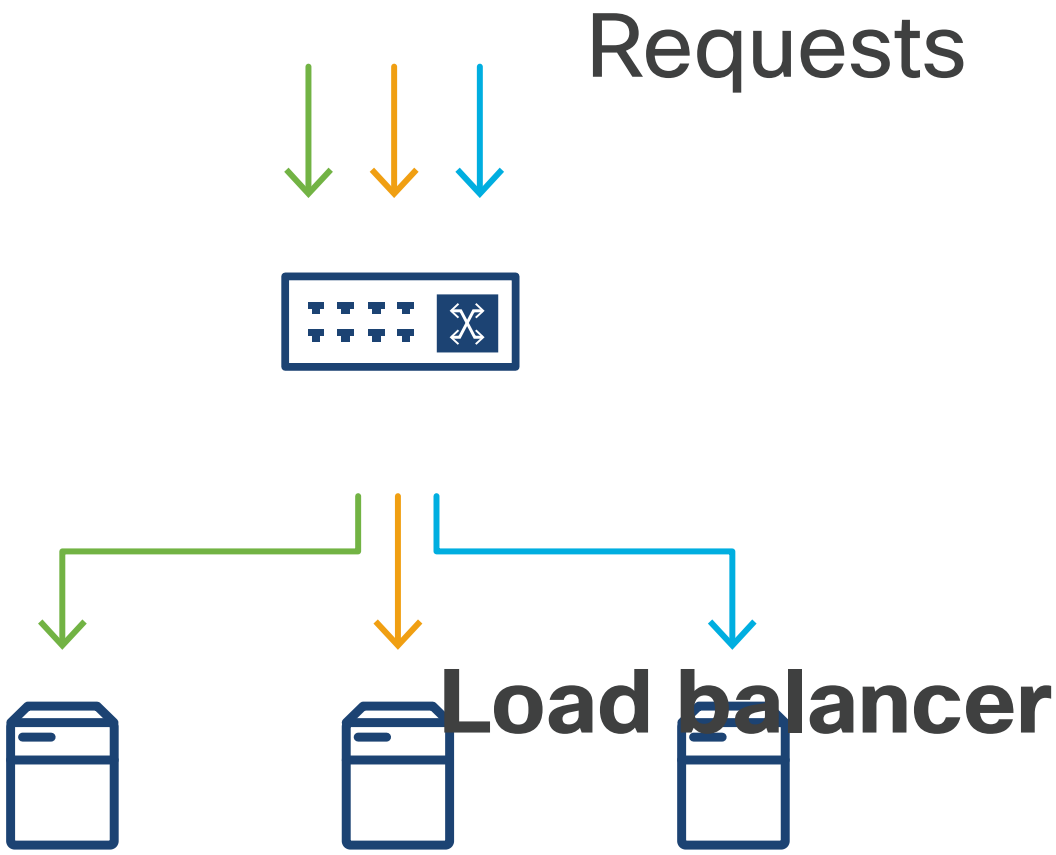
- Firewalls should keep *any* outside access to the untested application from occurring.
- Firewalls need to be configured so that the application can be appropriately tested. For example, if the application needs to access a development version of a database, the firewall rules will need to allow that.
- The environment should be as close a replica of production as possible in order to catch any firewall-related configuration issues quickly.

Note that firewalls do not just keep traffic from coming in; they can also be configured to keep traffic from getting out. For example, schools often have firewalls set up that keep students from accessing all but a small handful of educational sites using the school network.

6.4.3

Load Balancer

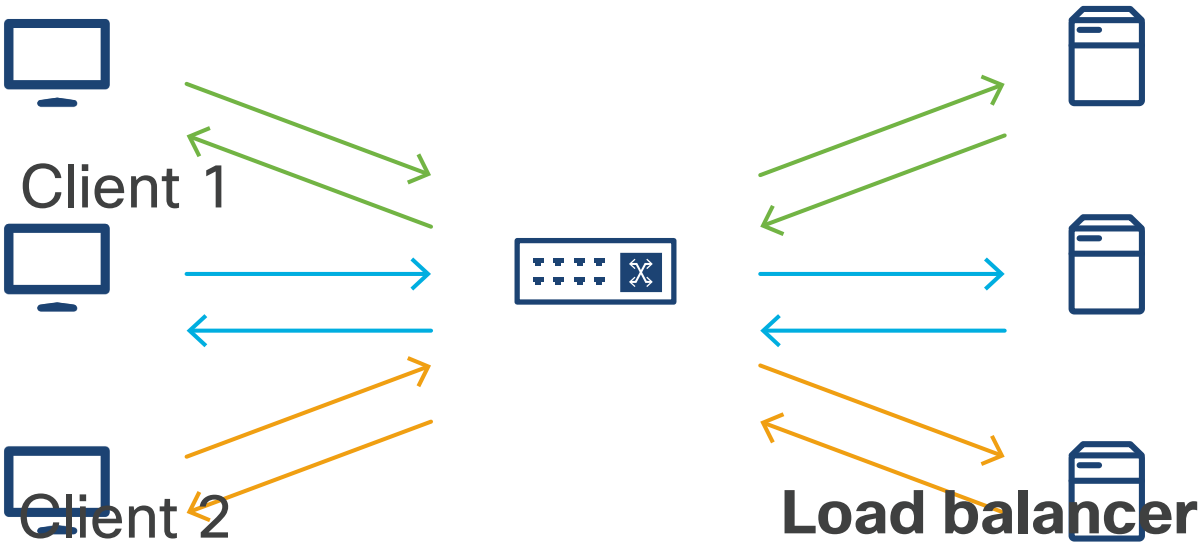
A load balancer does exactly what it says; it takes requests and “balances” them by spreading them out among multiple servers. For example, if you have 10 servers hosting your web application, requests will come first to the load balancer, which will then parcel them out among those 10 hosts.



A load balancer parcels out requests to different servers.

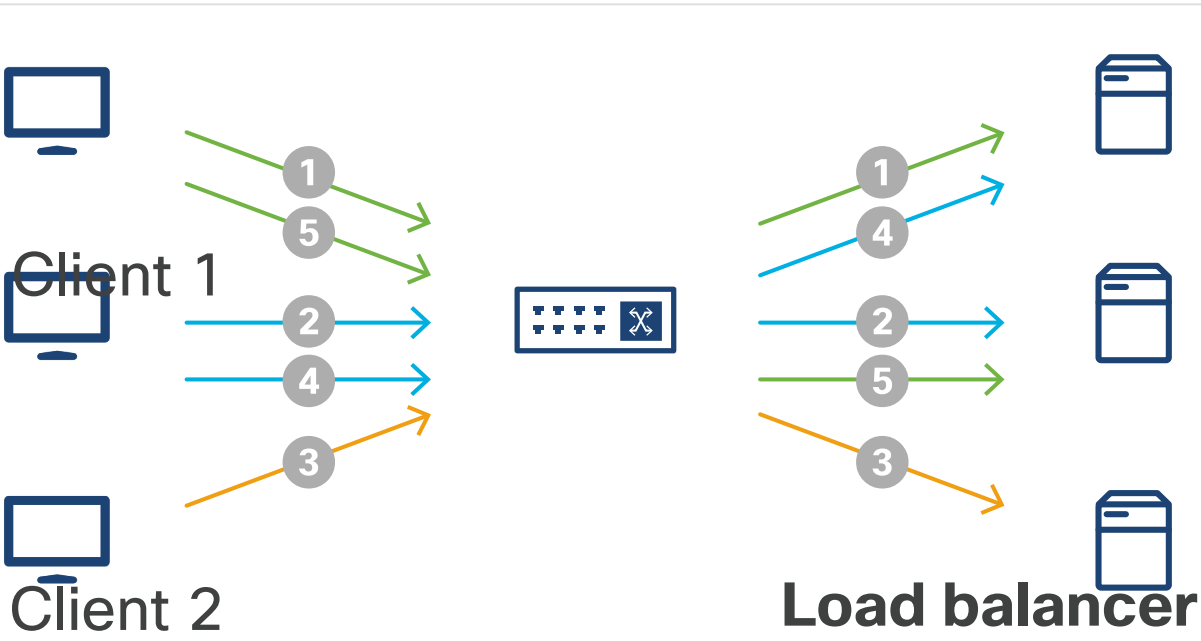
Load balancers can make their decisions on which servers should get a particular request in a few different ways:

Persistent sessions - If an application requires a persistent session, for example, a user needs to be logged in, the load balancer will send requests to the server handling that session.



Persistent sessions

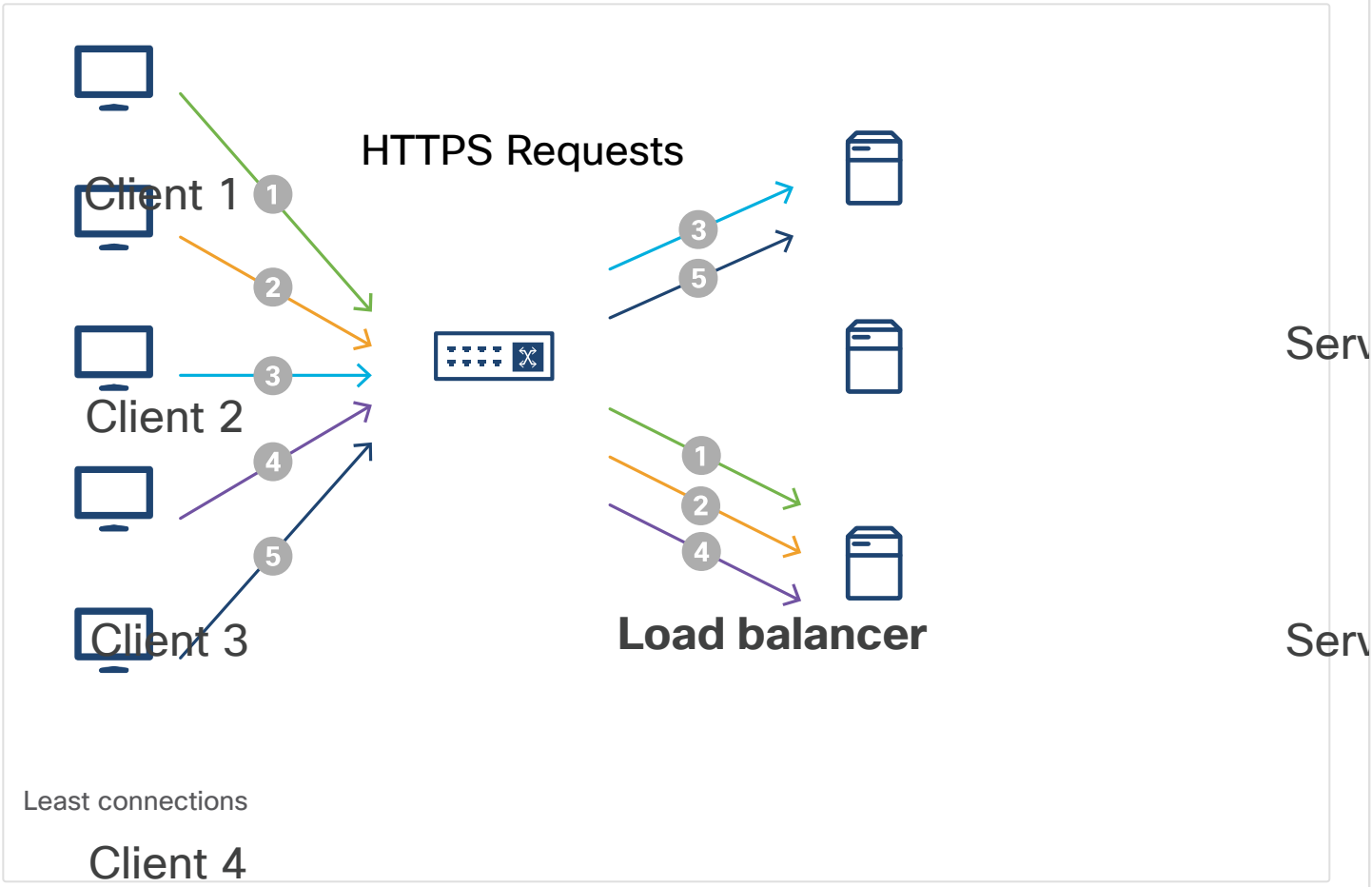
Round robin – With round robin load balancing, the server simply sends each request to the “next” server on the list.



Round robin

Least connections – Often it makes sense to send requests to the server that is the least “busy” – the least number of active connections. In the figure, Server 3 receives the first request because it is currently not handling any transactions. Server 3 receives the second request because the service has the least

number of active transactions. Server 1 and Server 3 both now have two active transactions, so load balancer will now use the round robin method. Server 1 receives the third request, Server 3 the fourth request and Server 1 the fifth request.

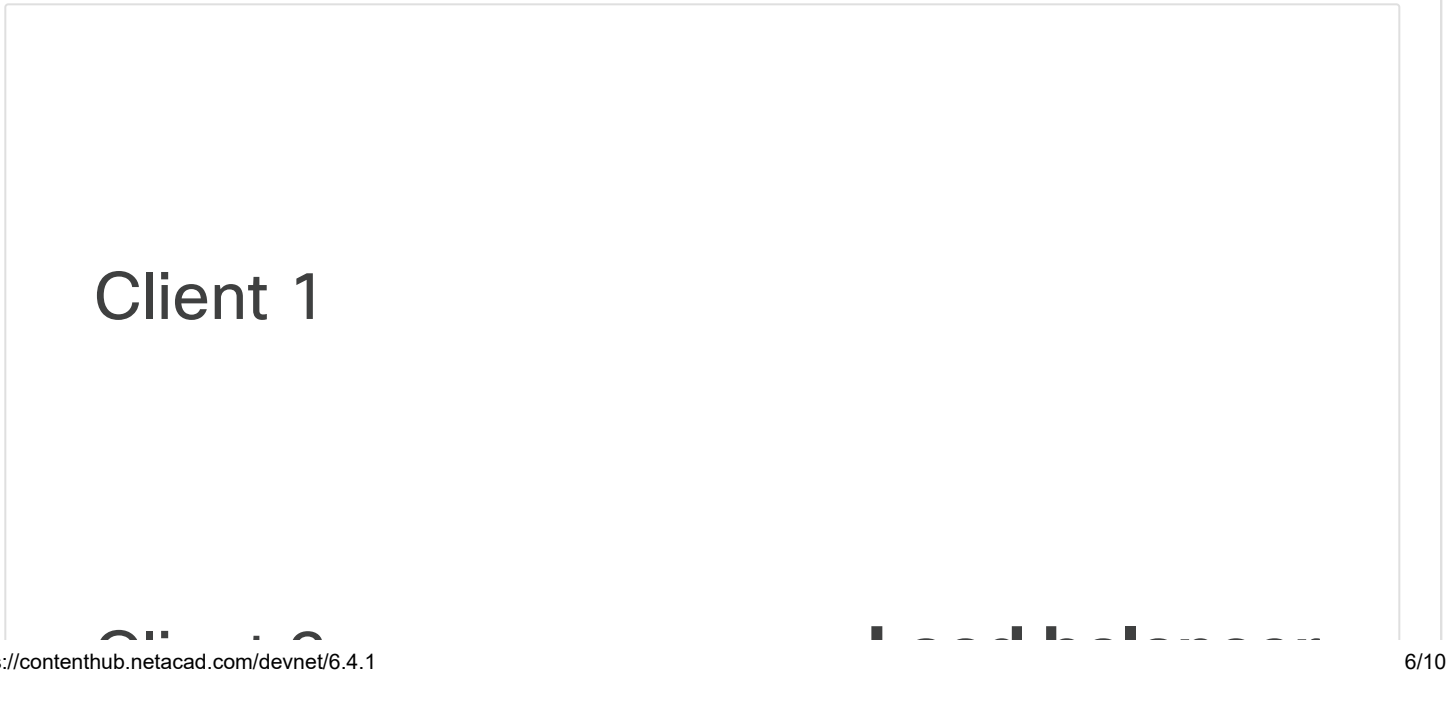


Least connections

Client 4

IP Hash - With this algorithm, the load balancer makes a decision based on a hash (an encoded value based on the IP address of the request). You can think of this as similar to when you attend an event and lines are formed for different stations based on the first letter of your last name. This is also a simple way to maintain consistent sessions.

Client 5



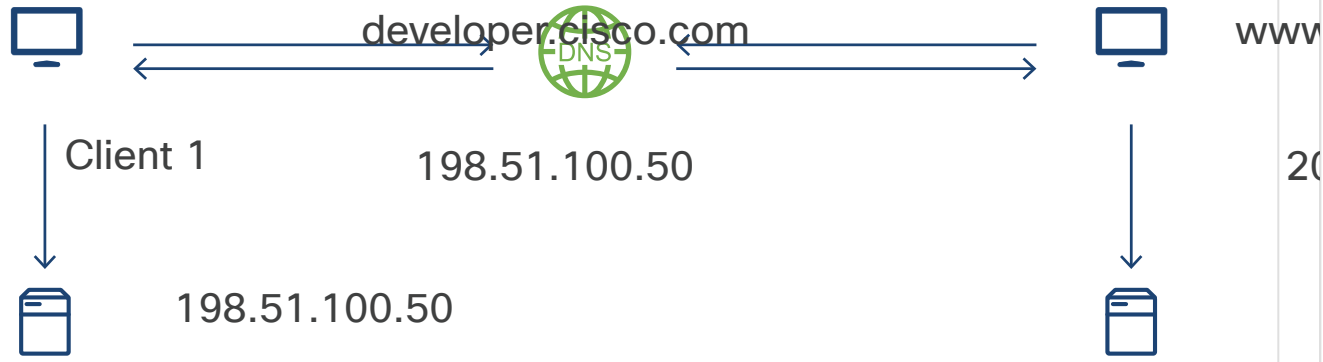
Other, more complicated algorithms can be used for deployment purposes. Some of these examples include:

1. **Blue-green deployment -** Recall that this kind of deployment applies changes to a new production environment (blue) rather than making the changes on the existing production environment (green). A load balancer sends traffic to the blue environment when it is ready, and if issues arise, the load balancer can send traffic back to the green environment and changes can be rolled back.
2. **Canary deployment -** This deployment starts by diverting a small fraction of your traffic to the blue environment. A load balancer can then increase the amount of traffic diverted to the blue environment until issues are detected and traffic goes back to the old environment, or all servers and users are on the new environment, and the old one is retired or used for the next push.

6.4.4

DNS

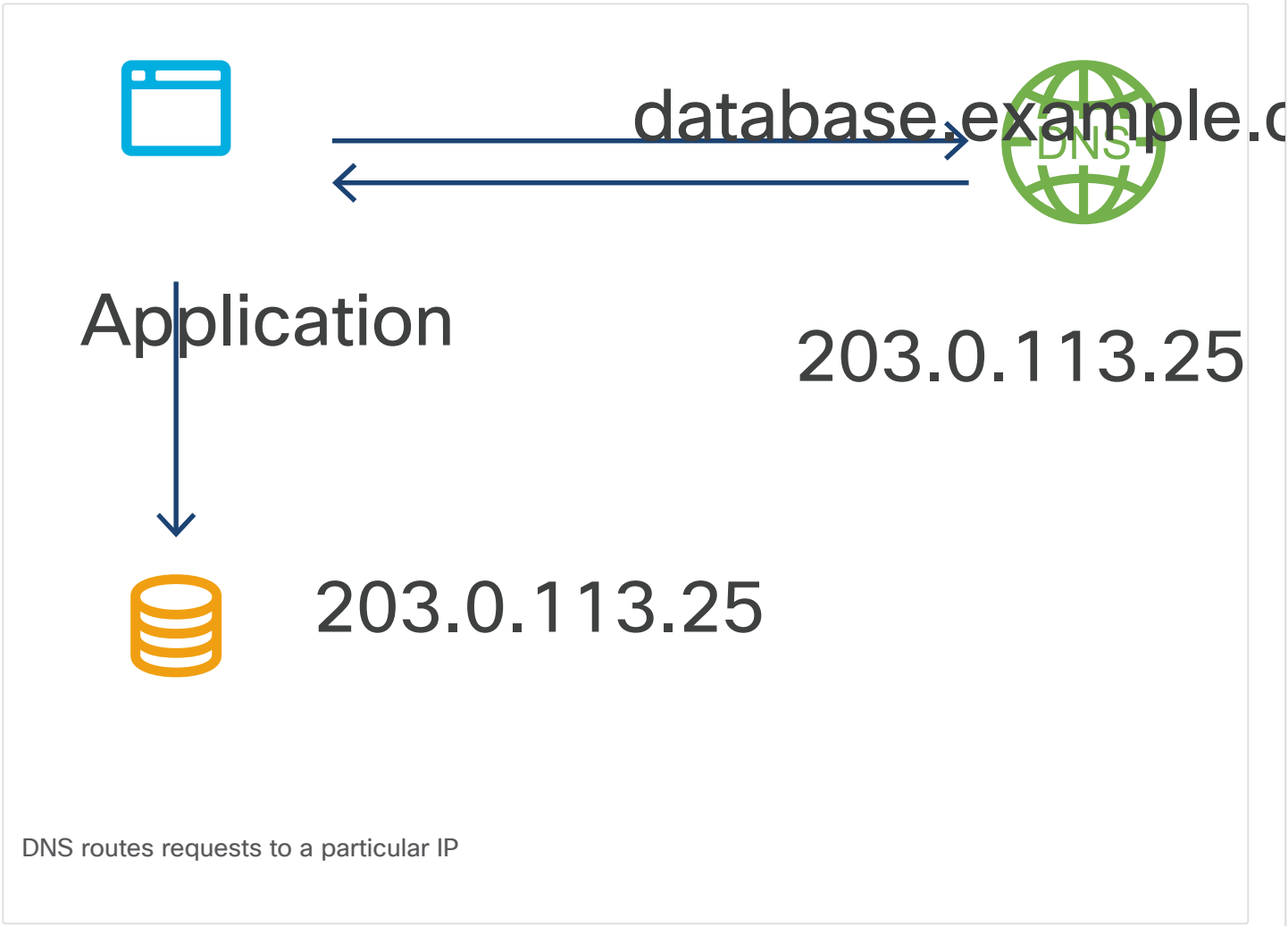
DNS, or the Domain Name System, is how servers on the internet translate human-readable names (such as developer.cisco.com or www.example.com) into machine-routable IP addresses such as 74.125.157.99 (for Google) or 208.80.152.201 (for Wikipedia). These IP addresses are necessary to actually navigate the internet.



DNS translates hostnames into (made-up) IP addresses.

developer.cisco.com

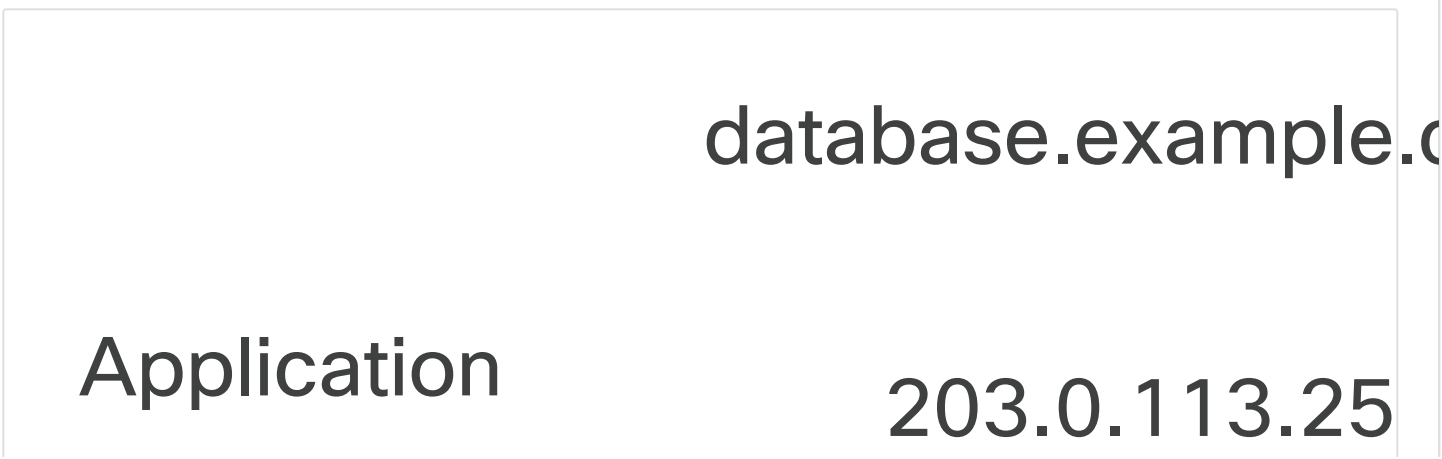
In software deployment, this system is beneficial because you can change the meaning of these addresses. In this example, the application is coded to look for the database at database.example.com:5000, which lives at the IP address of 203.0.113.25.

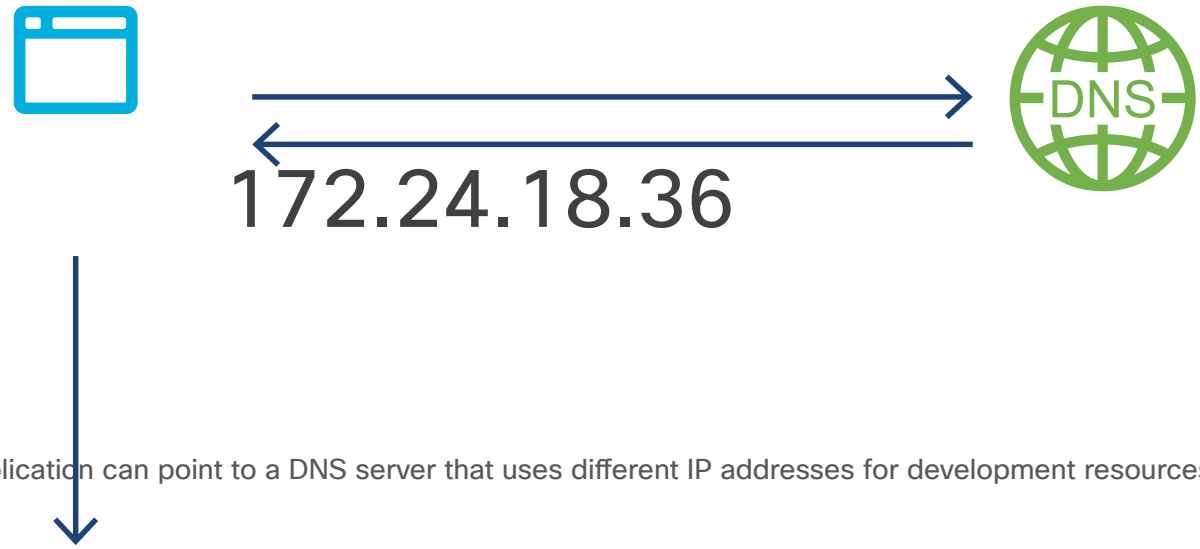


DNS routes requests to a particular IP

In another example, you might create a development version of the application, and you would want it to hit a development version of the database, which lives at 172.24.18.36.

You can set the development machine to use a DNS server that lists database.example.com as 172.24.18.36. You can test the application against the test database without actually making any changes to the application.





An application can point to a DNS server that uses different IP addresses for development resources.

Another way to use DNS as part of software deployment is to emulate some of the functions that might be performed by a load balancer. Do this by changing the IP address of the target server when you are ready to go "live". (This is not necessarily a good option because DNS changes can take a day or more to propagate through the internet at large.)

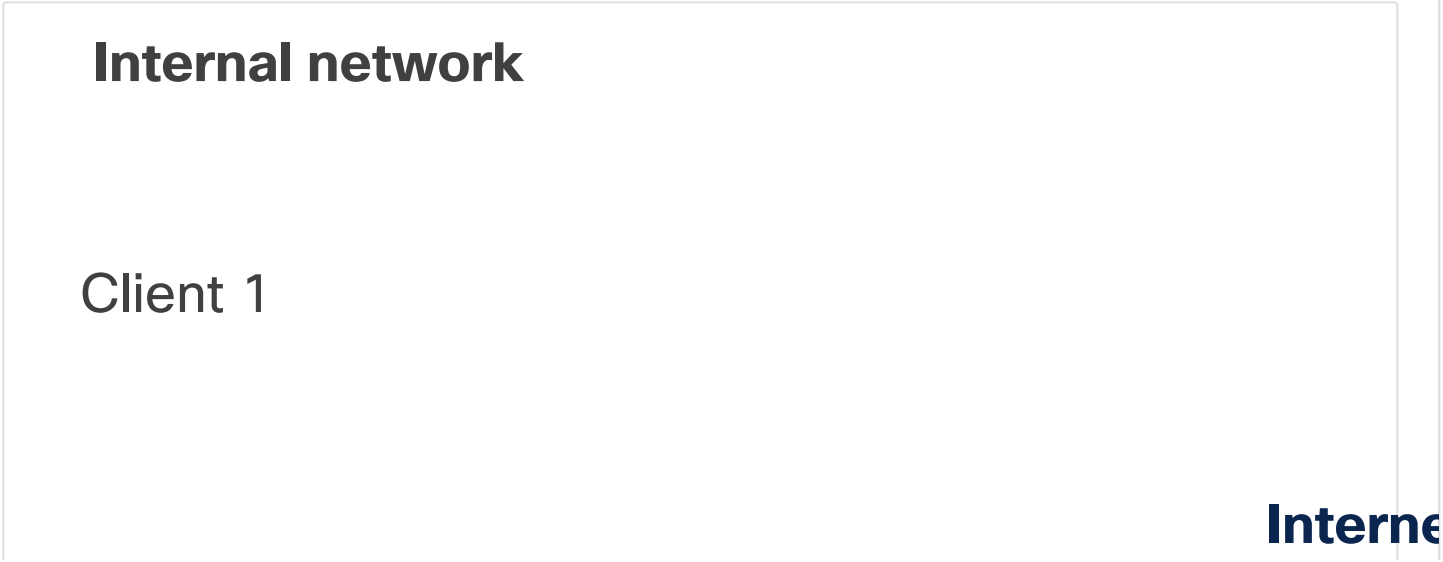
Database

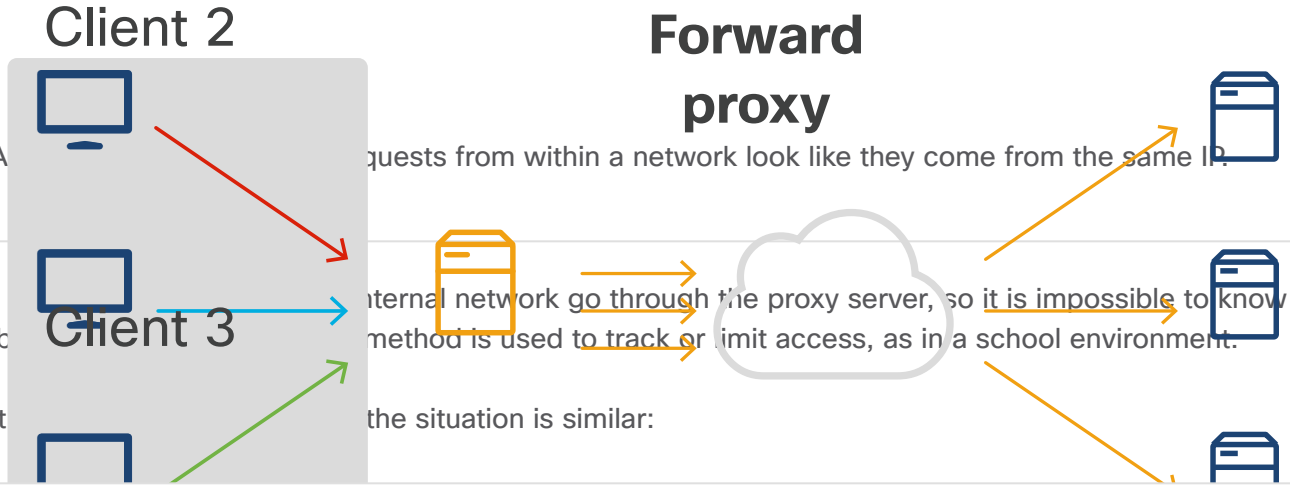
6.4.5

Reverse Proxy

A reverse proxy is similar to a regular proxy: however, while a regular proxy works to make requests from multiple computers look like they all come from the same client, a reverse proxy works to make sure responses look like they all come from the same server.

Here is an example of a forward proxy:

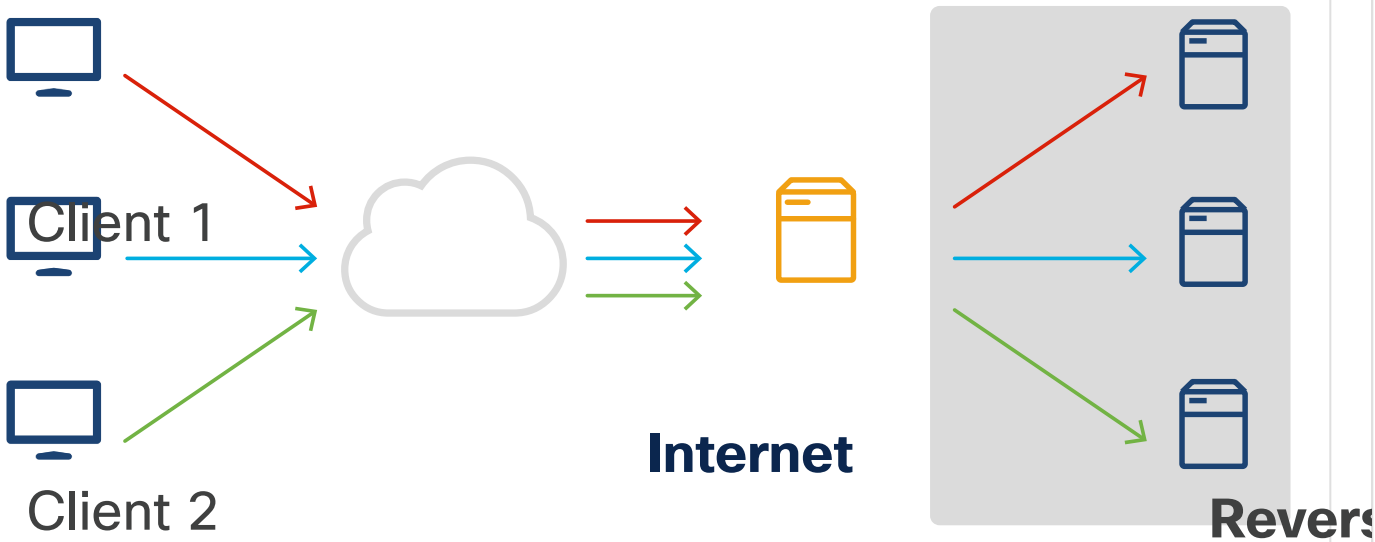




A quests from within a network look like they come from the same IP.

All nternal network go through the proxy server, so it is impossible to know what is b method is used to track or limit access, as in a school environment.

In t the situation is similar:



A reverse proxy enables requests to a single IP to be parceled out to different destinations inside the network.

All requests to the network come to the proxy, where they are evaluated and sent to the appropriate internal server for processing. Like a forward proxy, a reverse proxy can evaluate traffic and act accordingly. In this way, it is similar to, and can be used as, a firewall or a load balancer.

Because it is so much like these functions, Reverse Proxy can also be used for software deployment in similar ways.

6.3
Continuous Integration/Continuous Deploym...

6.5
Securing Applications