Contents

[IP classification: 1](#_Toc51321225)

[Subnets : 2](#_Toc51321226)

[How to Subnet: 4](#_Toc51321227)

[Reverse Proxy Server 7](#_Toc51321228)

[Load Balancer: 7](#_Toc51321229)

[DNS Round Robin (rarely used) 8](#_Toc51321230)

[L3/L4 Load Balancer 8](#_Toc51321231)

[L7 Load Balancer 8](#_Toc51321232)

[DNS 9](#_Toc51321233)

[DNS Record Types 9](#_Toc51321234)

[A and AAAA 9](#_Toc51321235)

[CNAME 9](#_Toc51321236)

[MX 10](#_Toc51321237)

[NS 11](#_Toc51321238)

[PTR 11](#_Toc51321239)

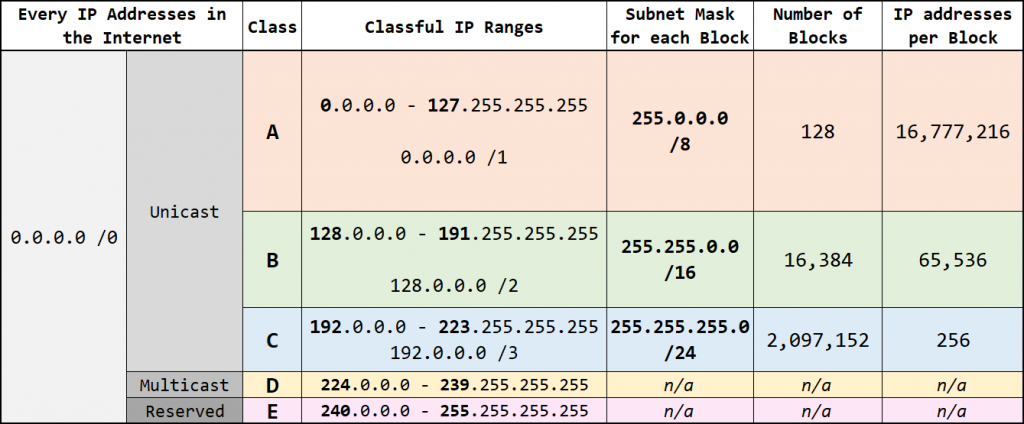
[SOA 12](#_Toc51321240)

[TXT 12](#_Toc51321241)

# IP classification:

IP address are assigned using two strategies

1. Classful: Classful addressing is how the early Internet was formed. IP assignments were given on the Octet boundaries:



However, this led to a lot of wasted IP addresses. If, for instance, you only needed 300 IP addresses, a Class C would not suffice, so you would end up with a Class B and nearly 60,000 IP addresses would be wasted.

Three popular private IP ranges are:

|  |  |  |
| --- | --- | --- |
| Class | IP Range | Purpose |
| A | 10.0.0.0/8 | Used for local communications within a private network |
| B | 172.16.0.0/12 | Used for local communications within a private network |
| C | 192.168.0.0/16 | Used for local communications within a private network |

1. Classless (CIDR)

With **Classless Inter-Domain Routing** (**CIDR**), IP assignments are not limited to the three classes. The whole unicast range (any IP address with a first octet of 0 – 223) can be allocated in any size block. In effect, the whole concept of IP address classes is done away with entirely.

## Subnets :

( <https://www.youtube.com/watch?v=XQ3T14SIlV4> )

**Subnetting allows you to create smaller network (sub networks; subnets) inside a large network** by borrowing bits from the Host ID portion of the address. We can use those borrowed bits to create additional networks, resulting in smaller-sized networks.

Imagine I want to build a network that will support up to 30 devices in different segments. Without subnetting, I will need four (4) Class C networks to support this design. For example:

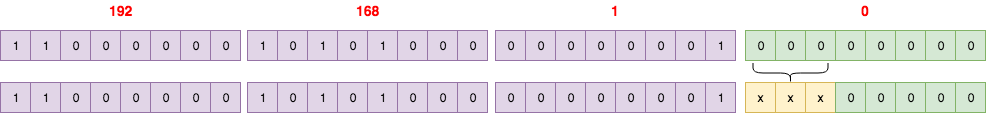
* Network #1: 192.168.1.0
* Network #2: 192.168.2.0
* Network #3: 192.168.3.0
* Network #4: 192.168.4.0

Each of these networks will support 254 IP addresses leading to a wastage of (254 \* 4) – (30 \* 4) IP addresses i.e. 896 IP addresses!

If you look at the design requirement of 30 hosts per network, you will discover that I only need 5 bits in the host ID portion of a Class C network to satisfy my requirement.

Usable Ip addresses

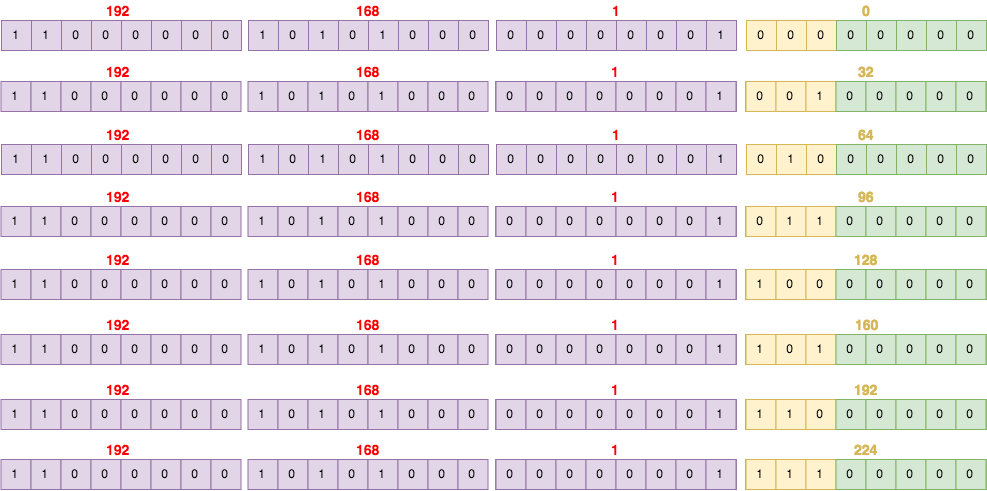
This means I still have 3 bits unused and with subnetting, I can use those three bits to create smaller networks. For this example, let’s take the 192.168.1.0 network:



By borrowing 3 bits, I can create 8 subnets:

1. 192.168.1.0
2. 192.168.1.32
3. 192.168.1.64
4. 192.168.1.96
5. 192.168.1.128
6. 192.168.1.160
7. 192.168.1.192
8. 192.168.1.224

These subnet addresses probably look weird to you – they look like normal IP addresses. However, looking at them in their binary form makes things clearer:



With subnetting, not only have we used only one Class C network, we have created 8 subnets from that network, each one supporting up to 30 hosts! We can use 4 of these subnets for our network and reserve the remaining 4 subnets for future expansion. This results in great waste reduction – from 896 *wasted* IP addresses to 120 *reserved* IP addresses.

**A subnet mask is the representation of the network portion of an address** in a Subnet.

**A switch or layer 2 devices looks for subnet mask to route traffic within the subnet or outside it.**

<https://www.youtube.com/watch?v=yLeuGOOrUvo>

<https://www.youtube.com/watch?v=Xb1JA5CIssI>

<https://www.ittsystems.com/introduction-to-subnetting/>

## How to Subnet:

Subnetting problem comes in two parts i.e either you will have to create subnets that can accommodate certain number of devices or directly create X number of subnet.

**Example 1: Directly create X number of subnet**

( <https://www.youtube.com/watch?v=ecCuyq-Wprc> )

Here is the network ID **192.168.4.0/24**, please create **3** separate networks or subnets for a coffee shop: Sunny Cafe.

List each of new **network ID**, **subnet mask**, **host ID range**, **# of usable hosts**, and **broadcast ID**.

Step 1: Identify number of **non-reserved bit** i.e 32-24 = 8

Step 2: Since we need to block that many IP bits so for 3 subnets we would require two bits i.e 2^2 = 4 . While one subnet will never be used but this is the best possible combination.

So **subnet mask: /26**

Now left number of bits would be 6 and hence **no. of usable host will be: 2^6-2 = 62**

Note: Subtracting two from each subnet mask as each subnet has Network ID (The first IP of subnet) and Broadcast ID ( Last IP of subnet) . **Cloud solution normally reserve 5 IPs instead of two. Hence usable will be what is left from reserved IPs in a subnet mask.**

**For multiplication help, remember this table:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 124 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

Step 3: Now for finding host range and network ID and broadcast ID

Since the IP was /24 CIDR, last octet of IP will start from 0 as :

11111111.11111111.11111111.**00000000**

Also each subnet has 64 hosts, Hence network ID becomes 192.168.4.0 and Broadcast ID becomes **192.168.4.63**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Network ID | Broadcast ID | Host range | Subnet Mask | #Usable Hosts |
| 192.168.4.0 | **192.168.4.63** | **192.168.4.1-192.168.4.62** | /26 | 62 |
| 192.168.4.64 | **192.168.4.127** | **192.168.4.65-192.168.4.126** | /26 | 62 |
| 192.168.4.128 | **192.168.4.191** | **192.168.4.129-192.168.4.190** | /26 | 62 |
| 192.168.4.192 | **192.168.4.255** | **192.168.4.193-192.168.4.254** | /26 | 62 |

**Example 2: Suppose I have a class B ID: 172.16.0.0/16 and now we want 4 subnets**

**(** [**https://www.youtube.com/watch?v=wuIdYxaV46Y**](https://www.youtube.com/watch?v=wuIdYxaV46Y) **)**

Step 1 : Identify non reserved bits i.e 32-16 = 16 bit.

Step 2 : Number of bits to be reserved for subnet : 2^2= 4 i.e 2 bits. Hence our **subnet mask will be /18**

Number of usable host per subnet will be 2^14-2 = 16384-2 = **16382** hosts ( Two being reserved for Network and Broadcast ID )

Step 3: Now for finding host range, network ID and broadcast ID

Since the IP cidr is /16 hence the first IP of subnet will start from last two octets or non-reserved bits.

11111111.11111111.00000000.00000000 i.e 172.16.0.0

Also since each subnet has 2^14 hosts that would translate as

|  |  |  |  |
| --- | --- | --- | --- |
| Network Mask | Network ID | Network Mask | Broadcast ID |
| XXXXXXXX.XXXXXXXX.**00**000000.000000 | 172.16.0.0 | XXXXXXXX.XXXXXXXX.00111111.111111 | 172.16.63.255 |
| XXXXXXXX.XXXXXXXX.**01**000000.000000 | 172.16.64.0 (2^6) | XXXXXXXX.XXXXXXXX.01111111.111111 | 172.16.127.255 |
| XXXXXXXX.XXXXXXXX.**10**000000.000000 | 172.16.128.0 (2^7) | XXXXXXXX.XXXXXXXX.10111111.111111 | 172.16.191.255 |
| XXXXXXXX.XXXXXXXX.**11**000000.000000 | 172.16.192.0  (2^7+2^6) | XXXXXXXX.XXXXXXXX.11111111.111111 | 172.16.255.255 |

To summarize:

If one is give /24 type networks use following table for quick translation of Class C subnets :

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Number of Subnets** | **1** | **2** | **4** | **8** | **16** | **32** | **64** | **128** | **256** |
| **Hosts** | **256** | **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| **Subnet Mask** | **/24** | **/25** | **26** | **/27** | **/28** | **/29** | **/30** | **/31** | **/32** |

Similarly, if one is give /16 type of network then one can use the following Class B subnets:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Number of Subnets** | **1** | **2** | **4** | **8** | **16** | **32** | **64** | **128** | **256** | **512** | **1024** | **2048** | **4096** | **8192** | **163184** | **32768** | **65536** |
| **Hosts** | **65536** | **32768** | **163184** | **8192** | **4096** | **2028** | **1024** | **512** | **256** | **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| **Subnet Mask** | **/16** | **/17** | **/18** | **/19** | **/20** | **/21** | **/22** | **/23** | **/24** | **25** | **/26** | **/27** | **/28** | **/29** | **/30** | **/31** | **/32** |

Note : Azure and AWS reserve 5 IPs per subnet so in that case subtract 5 from number of hosts available per subnet for calculating usable hosts.

Azure reserves 5 IP addresses within each subnet. These are x.x.x.0-x.x.x.3 and the last address of the subnet. x.x.x.1-x.x.x.3 is reserved in each subnet for Azure services.

* x.x.x.0: Network address
* x.x.x.1: Reserved by Azure for the default gateway
* x.x.x.2, x.x.x.3: Reserved by Azure to map the Azure DNS IPs to the VNet space
* x.x.x.255: Network broadcast address

# Reverse Proxy Server

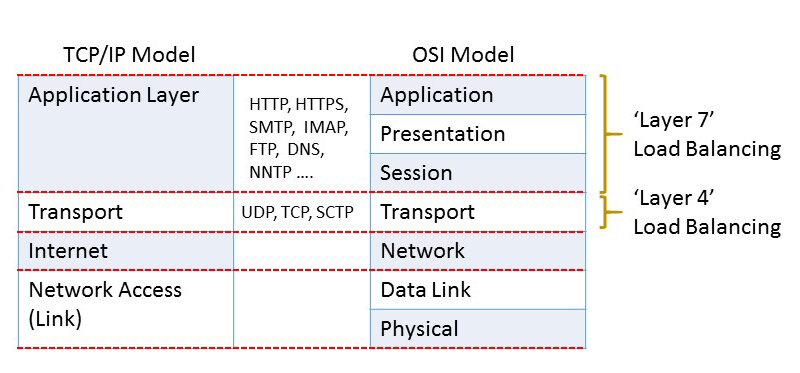
A proxy server is a go‑between or intermediary server that forwards requests for content from multiple clients to different servers across the Internet. A **reverse proxy server** is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate backend server. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.

Common uses for a [reverse proxy server](https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/) include:

* **Load balancing** – A reverse proxy server can act as a “traffic cop,” sitting in front of your backend servers and distributing client requests across a group of servers in a manner that maximizes speed and capacity utilization while ensuring no one server is overloaded, which can degrade performance. If a server goes down, the [load balancer](https://www.nginx.com/solutions/adc/) redirects traffic to the remaining online servers.
* **Web acceleration** – Reverse proxies can compress inbound and outbound data, as well as cache commonly requested content, both of which speed up the flow of traffic between clients and servers. They can also perform additional tasks such as SSL encryption to take load off of your web servers, thereby [boosting their performance](https://www.nginx.com/resources/glossary/web-acceleration/).
* **Security and anonymity** – By intercepting requests headed for your backend servers, a reverse proxy server protects their identities and acts as an additional defence against security attacks. It also ensures that multiple servers can be accessed from a single record locator or URL regardless of the structure of your local area network.

<https://www.nginx.com/resources/glossary/reverse-proxy-server/>

# Load Balancer:



Types of Load Balancer:

## DNS Round Robin (rarely used)

clients get a randomly-ordered list of IP addresses.  
pros: easy to implement and free  
cons: hard to control and not responsive, since DNS cache needs time to expire

## L3/L4 Load Balancer

traffic is routed by IP address and port. L3 is network layer (IP). L4 is session layer (TCP).  
pros: better granularity, simple, responsive.

Today the term “Layer 4 load balancing” most commonly refers to a deployment where the load balancer’s IP address is the one advertised to clients for a web site or service (via DNS, for example). As a result, clients record the load balancer’s address as the destination IP address in their requests.

## L7 Load Balancer

traffic is routed by what is inside the HTTP protocol. L7 is application layer (HTTP).

Layer 7 load balancers base their routing decisions on various characteristics of the HTTP header and on the actual contents of the message, such as the URL, the type of data (text, video, graphics), or information in a cookie.

Taking into consideration so many more aspects of the information being transferred can make [Layer 7 load balancing](https://www.nginx.com/resources/glossary/layer-7-load-balancing) more expensive than Layer 4 in terms of time and required computing power, but it can nevertheless lead to greater overall efficiency. For instance, because a Layer 7 load balancer can determine what type of data (video, text, and so on) a client is requesting, you don’t have to duplicate the same data on all of the load-balanced servers. It uses buffering to offload slow connections from the upstream servers, which improves performance.

**A device that performs Layer 7 load balancing is often referred to as a** [***reverse‑proxy server***](https://www.nginx.com/resources/glossary/reverse-proxy-server/)**.**

<https://www.nginx.com/resources/glossary/layer-7-load-balancing>

# DNS

## DNS Record Types

<https://www.linode.com/docs/networking/dns/dns-records-an-introduction/>

<https://www.youtube.com/watch?v=e2xLV7pCOLI>

### A and AAAA

An A record points your domain or subdomain to your Linode’s IP address, which allows web traffic to reach your Linode. This is the core function of DNS. A typical A record looks like either of the following:

example.com A 12.34.56.78

hello.example.com A 12.34.56.78

You can point different subdomains to different IP addresses. If you want to point every subdomain of example.com to your Linode’s IP, you can use an asterisk (**\***) as your subdomain:

\*.example.com A 12.34.56.78

An AAAA record is just like an A record, but for IPv6 IP addresses. A typical AAAA record looks like the following:

example.com AAAA 0123:4567:89ab:cdef:0123:4567:89ab:cdef

### CNAME

A CNAME record or Canonical Name record matches a domain or subdomain to a different domain. With a CNAME record, DNS lookups use the target domain’s DNS resolution as the alias’s resolution. Here’s an example:

alias.com CNAME example.com.

example.com A 12.34.56.78

With this setup, when alias.com is requested, the initial DNS lookup will find the CNAME entry with the target of example.com. A new DNS lookup will be started for example.com, which will find the IP address 12.34.56.78. Finally, visitors to alias.com will be directed to 12.34.56.78.

CNAME records exist so that domains can have aliases. Some mail servers handle mail oddly for domains with CNAME records, so you should not use a CNAME record for a domain that gets email. Likewise, MX records cannot reference CNAME-defined hostnames. The target domain for a CNAME record should also have a normal A-record resolution. Chaining or looping CNAME records is not recommended.

**Note**

In some cases, a CNAME record can be an effective way to redirect traffic from one domain to another while keeping the same URL. However, keep in mind that a CNAME record does not function the same way as a URL redirect. A CNAME record directs web traffic for a particular domain to the target domain’s IP address. Once the visitor reaches that IP address, the web server’s configuration will determine how the domain is handled. If that domain is not configured on the server, the server will simply display its default web page (if any). This may or may not be the web page for the target domain in the CNAME record, depending on how the server is configured.

### MX

An MX record or mail exchanger record sets the mail delivery destination for a domain or subdomain. A typical MX record looks like the following:

example.com MX 10 mail.example.com.

mail.example.com A 12.34.56.78

The above records direct mail for example.com to the mail.example.com server. The target domain (mail.example.com above) needs to have its own A record that resolves to your Linode. An MX record should ideally point to a domain that is also the [hostname](https://www.linode.com/docs/getting-started/#set-the-hostname) for its server.

Your MX records don’t necessarily have to point to your Linode. If you’re using a third-party mail service like [Google Apps](https://www.linode.com/docs/email/using-google-apps-for-email/), you should use the MX records they provide.

Priority is another component of MX records. This is the number written between the record type and the target server (10 in the example above). Priority allows you to designate a fallback server (or servers) for mail for a particular domain. Lower numbers have a higher priority. Here’s an example of a domain that has two fallback mail servers:

example.com MX 10 mail\_1.example.com

example.com MX 20 mail\_2.example.com

example.com MX 30 mail\_3.example.com

In this example, if mail\_1.example.com is down, mail will be delivered to mail\_2.example.com. If mail\_2.example.com is also down, mail will be delivered to mail\_3.example.com.

### NS

NS records or name server records set the nameservers for a domain or subdomain. The primary nameserver records for your domain are set both at your registrar and in your zone file. Typical nameserver records (you need at least two) look like this:

example.com NS ns1.linode.com.

example.com NS ns2.linode.com.

example.com NS ns3.linode.com.

example.com NS ns4.linode.com.

example.com NS ns5.linode.com.

The nameservers you designate at your registrar then carry the zone file for your domain.

You can also set up different nameservers for any of your subdomains. Subdomain NS records get configured in your primary domain’s zone file. For example, if you’re using Linode’s nameservers, you could configure separate NS records in your Linode zone file for the subdomain mail.example.com as shown below:

mail.example.com NS ns1.nameserver.com

mail.example.com NS ns2.nameserver.com

Primary nameservers get configured at your registrar and secondary subdomain nameservers get configured in the primary domain’s zone file. The order of NS records does not matter. DNS requests are sent randomly to the different servers, and if one host fails to respond, another one will be queried.

### PTR

A PTR record or pointer record matches up an IP address to a domain or subdomain, allowing reverse DNS queries to function. It performs the opposite service an A record does, in that it allows you to look up the domain associated with a particular IP address, instead of vice versa.

PTR records are usually set with your hosting provider. They are not part of your domain’s zone file. This means that you’ll always set reverse DNS for your Linodes in the Linode Manager, even if your nameservers are elsewhere. Likewise, if you have servers somewhere else but are using Linode’s nameservers, you will still have to set up your PTR records with your hosting provider.

As a prerequisite for adding a PTR record, you need to create a valid, live A or AAAA record that points the desired domain to that IP. If you want an IPv4 PTR record, point the domain or subdomain to your Linode’s IPv4 address. If you want an IPv6 PTR record, point the domain to your Linode’s IPv6 address. Beyond that, IPv4 and IPv6 PTR records work the same way.

For instructions on setting up reverse DNS on your Linode, see our [Reverse DNS](https://www.linode.com/docs/networking/dns/configure-your-linode-for-reverse-dns/#setting-reverse-dns) guide.

**Note**

It’s possible to have different IPs (including both IPv4 and IPv6 addresses) that have the same domain set for reverse DNS. To do this, you will have to configure multiple A or AAAA records for that domain that point to the various IPs.

### SOA

An SOA record or Start of Authority record labels a zone file with the name of the host where it was originally created. Next, it lists the contact email address for the person responsible for the domain. There are also various numbers, which we’ll get into in detail in a moment. First, here’s a typical SOA record:

@ IN SOA ns1.linode.com. admin.example.com. 2013062147 14400 14400 1209600 86400

**Note**

The administrative email address is written with a period (**.**) instead of an **@** symbol.

Here’s what the numbers mean:

* **Serial number**: The revision number for this domain’s zone file. It changes when the file gets updated.
* **Refresh time**: The amount of time (in seconds) a secondary DNS server will keep the zone file before it checks for changes.
* **Retry time**: The amount of time a secondary DNS server will wait before retrying a failed zone file transfer.
* **Expire time**: The amount of time a secondary DNS server will wait before expiring its current zone file copy if it cannot update itself.
* **Minimum TTL**: The minimum amount of time other servers should keep data cached from this zone file.

The single nameserver mentioned in the SOA record is considered the primary master for the purposes of Dynamic DNS and is the server where zone file changes get made before they are propagated to all other nameservers.

### TXT

A TXT record or text record provides information about the domain in question to other resources on the internet. It’s a flexible type of DNS record that can serve many different purposes depending on the specific contents. One common use of the TXT record is to create an [SPF record](https://www.linode.com/docs/networking/dns/dns-records-an-introduction/#spf) on nameservers that don’t natively support SPF. Another use is to create a [DKIM record](https://www.linode.com/docs/networking/dns/dns-records-an-introduction/#dkim) for mail signing.