Engineer

CIDR

describir y enrutar bloques de direcciones IP con prefijos

Qué es

CIDR, siglas de Classless Inter-Domain Routing, es el esquema que describe un bloque de direcciones IP mediante una dirección inicial y el número de bits iniciales que comparten todas las direcciones del bloque. Por escrito es la barra habitual: 198.51.100.0/24 significa todas las direcciones cuyos primeros 24 bits coinciden con 198.51.100.0, es decir, 256 direcciones desde 198.51.100.0 hasta 198.51.100.255. El número situado después de la barra se llama longitud de prefijo, y el propio bloque se llama prefijo.

La palabra classless, sin clases, remite a lo que había antes. El plan original de IPv4 de RFC 791 tenía tres tamaños de red, y RFC 4632, la descripción actual de CIDR, explica por qué falló: la clase C, con 254 direcciones de host utilizables, era demasiado pequeña para una organización mediana, y la clase B, con hasta 65.534, demasiado grande; aun así, la clase B era la mejor opción disponible. Con la comercialización de la red a finales de la década de 1980, la demanda de clase B creció a un ritmo que parecía exponencial.

En enero de 1992, el grupo ROAD del IETF, Routing and Addressing, identificó tres problemas: el agotamiento del espacio de clase B, unas tablas de enrutamiento que superaban lo que podían gestionar los enrutadores y sus operadores, y el agotamiento futuro de IPv4. Se esperaba que los dos primeros fueran críticos entre 1993 y 1995. CIDR fue la respuesta a esos dos, propuesta primero en RFC 1338 en junio de 1992 y después en RFC 1519 en septiembre de 1993, al que RFC 4632 sustituyó en agosto de 2006. Nunca pretendió resolver el tercero, solo ganar tiempo.

En notación CIDR, RFC 4632 escribe un prefijo con la dirección habitual de cuatro partes, una barra y un número de 0 a 32 que indica los bits significativos. La máscara de red anterior expresa lo mismo con esos bits a uno y el resto a cero,

por lo que /24 equivale a 255.255.255.0. La única restricción que mantiene el RFC es que la máscara sea contigua, y dentro de ella un prefijo puede describir cualquier bloque cuyo tamaño sea una potencia de dos. La tabla del RFC va desde /32, una ruta de host de una dirección, pasando por /31 para un enlace punto a punto con dos, /24 con 256 y /16 con 65.536, hasta 0.0.0.0/0, las 4.294.967.296 direcciones, utilizado como ruta predeterminada.

La notación sobrevivió a la familia de direcciones para la que se creó: RFC 4291, que define la estructura de las direcciones IPv6, escribe los prefijos IPv6 de la misma forma, por lo que 2001:db8::/32 se lee igual que sus equivalentes IPv4.

Utilice el concepto en una decisión

Aplique «¿Qué es CIDR?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Wi-Fi, cortafuegos y acceso remoto configurados para trabajar desde cualquier lugar y permitir el acceso solo a las personas autorizadas.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Administración de 30 VPS Ubuntu Linux — 30 servidores Linux administrados con recuperación planificada

Administración de 30 VPS Ubuntu Linux Póngase en contacto →

Por qué importa

CIDR resolvió sus problemas originales mediante la agregación. Según RFC 4632, resumir rutas es la única forma ampliamente conocida de que una red de conmutación de paquetes necesite mantener menos información de enrutamiento. Un proveedor que dispone de un bloque grande y asigna direcciones a sus clientes a partir de partes pequeñas de él puede anunciar una sola ruta para el bloque completo, y el resto de internet no necesita nada más para llegar a todos los clientes que contiene. RFC 4632 añade que la propia asignación de direcciones tuvo que cambiar para hacerlo posible, y que la parte del antiguo espacio de clase C que no puede agregarse, conocida como swamp, o pantano, todavía contenía unas 15.000 entradas de enrutamiento en 2006.

La agregación funciona gracias a una regla que sigue cada enrutador. En palabras de RFC 4632: «El reenvío en internet se realiza según la coincidencia más larga». Cuando más de un prefijo cubre un destino, gana el más específico. Así sigue siendo accesible una organización que cambia de proveedor sin renumerar: el proveedor nuevo anuncia el pequeño prefijo propio de la organización, lo que el RFC describe como abrir un hueco en el agregado del proveedor anterior, y se

prefiere al agregado porque la coincidencia es más larga. Previsto para durar de tres a cinco años, CIDR ya se había desplegado ampliamente en 2005, según el RFC.

Pocos ingenieros configuran enrutamiento entre dominios, pero casi todos escriben prefijos: en una regla de cortafuegos que admite la red de una oficina, en una tabla de rutas de un servidor Linux y en WireGuard, donde la lista de prefijos AllowedIPs de cada par decide tanto qué direcciones de origen se aceptan de él como adónde se envía el tráfico destinado a ese par. En todos esos lugares, un solo dígito incorrecto después de la barra admite o excluye silenciosamente miles de hosts.

Cómo se aplica en la práctica

El módulo estándar `ipaddress` de Python comprende la notación y permite comprobar rápidamente los cálculos antes de introducirlos en una regla. La sesión siguiente es un ejemplo de este artículo, ejecutado en Python 3.15 con una dirección del rango de documentación 198.51.100.0/24:

```
>>> import ipaddress
>>> net = ipaddress.ip_network("198.51.100.0/24")
>>> net.netmask, net.num_addresses
(IPv4Address('255.255.255.0'), 256)
>>> for sub in net.subnets(new_prefix=26):
...     print(sub)
...
198.51.100.0/26
198.51.100.64/26
198.51.100.128/26
198.51.100.192/26
>>> ipaddress.ip_network("198.51.100.70/26")
Traceback (most recent call last):
  ...
ValueError: 198.51.100.70/26 has host bits set
>>> ipaddress.ip_network("198.51.100.70/26", strict=False)
IPv4Network('198.51.100.64/26')
>>> half = ipaddress.ip_network("198.51.100.128/25")
>>> ipaddress.ip_address("198.51.100.200") in half
True
>>> half.subnet_of(net)
True
```

Un /24 se divide en cuatro bloques /26 de 64 direcciones, cada uno de los cuales empieza en un múltiplo de 64. Solicitar 198.51.100.70/26 falla porque el bloque que contiene .70 empieza en .64 y los bits posteriores al 26 no son cero; con `strict=False`, el módulo los pone a cero y devuelve el bloque al que pertenece .70. Las dos últimas líneas son las preguntas que debería hacerse quien revise cualquier par de prefijos: si la dirección está dentro y si un bloque está completamente dentro del otro.

La segunda pregunta es la más importante en un filtro de paquetes, que no utiliza la coincidencia más larga. Lee las reglas en orden y se detiene en la primera que decide. El manual nft(8) indica que `accept` y `drop` terminan la evaluación de la cadena, que `add` añade una regla al final y que `insert` la coloca al principio. En este segundo ejemplo elaborado para el artículo, el objetivo es permitir que un /24 acceda a SSH y excluir su mitad superior:

```
add table inet filter
add chain inet filter input { type filter hook input priority filter; }
add rule inet filter input ip saddr 198.51.100.0/24 tcp dport 22 accept
add rule inet filter input ip saddr 198.51.100.128/25 tcp dport 22 drop
```

Ambas reglas parecen correctas y un listado muestra las dos. Pero todas las direcciones del /25 están también en el /24, por lo que la primera regla acepta esos paquetes y nunca se alcanza el descarte situado debajo. En enrutamiento, ganaría el prefijo más específico independientemente de su posición; en un filtro, solo gana si va primero. Intercambiar las dos líneas o añadir la regla más restringida con `insert` produce el resultado pretendido. `nft -a list ruleset` imprime cada regla con su identificador (handle), y tanto `add` como `insert` aceptan un identificador para situar una regla nueva junto a otra existente.

La entrada de portafolio Reglas de cortafuegos verificadas por su posición, de agosto de 2026, describe la misma trampa desde la operación real. Un host funcionó durante un año con un límite de frecuencia que ningún paquete alcanzaba porque estaba debajo de una regla más amplia que coincidía antes, y todas las herramientas que imprimían un resumen mostraban el límite correctamente. Las comprobaciones leen ahora la lista numerada de reglas y la cadena activa del filtro y verifican su orden. Esa página no dice cómo estaba escrita la regla más amplia ni si intervenían prefijos de direcciones, por lo que las líneas anteriores ilustran el principio, no las reglas de ese host.

Dónde surgen los problemas

Bits de host en la dirección. 198.51.100.70/26 es una dirección con máscara, no un bloque. Python la rechaza de forma predeterminada, pero con `strict=False` la redondea hacia abajo a 198.51.100.64/26, cuyos hosts quizá no sean los que pretendía incluir quien escribió la regla. Escriba la dirección de red del bloque y analice en modo estricto para que el error aparezca como tal.

Una dirección sin máscara donde se pretendía un bloque. La documentación de Python indica que una red sin máscara se interpreta como /32, por lo que introducir la dirección de una oficina sin su prefijo admite un solo host. Escriba siempre la longitud del prefijo, incluso /32, para que se pueda distinguir una ruta de host de una barra olvidada.

Un prefijo amplio antes de otro más restringido. Como muestra el ejemplo, un filtro que decide en la primera coincidencia permite que una regla anterior oculte las posteriores cuyo prefijo contiene. Coloque primero las reglas más específicas y, al generar reglas, compruebe cada par con una prueba de inclusión como `subnet_of` u `overlaps` y haga fallar la compilación si una regla posterior es inalcanzable.

Una longitud de prefijo unos bits demasiado corta. Cada bit eliminado duplica el bloque, de modo que escribir /16 en lugar de /24 multiplica por 256 el número de direcciones admitidas, y /0 admite todas. RFC 4632 documenta que errores de operación y otros sucesos hicieron que en el pasado se observaran rutas /1 y /2 en algunas redes. Revise cualquier regla cuyo prefijo sea más corto que el bloque que posee y reserve 0.0.0.0/0 y ::/0 para reglas que deban coincidir con todo.

Contar hosts como direcciones. Un /24 tiene 256 direcciones, pero en una red ordinaria la primera es la dirección de red y la última la de difusión, por lo que `hosts()` de Python devuelve 254. En enlaces punto a punto, ese desperdicio se acumula: RFC 3021 señala que un /30 gasta cuatro direcciones en dos hosts y permite utilizar en su lugar un /31, lo que ahorra 1.000 direcciones en 500 enlaces. Dimensione el bloque para los hosts que debe contener más esas direcciones reservadas y utilice /31 solo en equipos compatibles con RFC 3021.

Un agregado sin ruta de descarte. Un enrutador que anuncia un bloque resumido pero ha perdido uno de los prefijos menores que contiene puede devolver el tráfico de esa parte ausente por una ruta menos específica, como la predeterminada, y crear un bucle de reenvío. RFC 4632 exige que el enrutador

que genera el agregado descarte los paquetes que coincidan con él y no con una ruta más específica, lo que en la práctica supone instalar una ruta de descarte o nula para cada agregado que anuncia.

Listas de permitidos que solo cubren IPv4. En nftables, `ip saddr` lee la cabecera IPv4 e `ip6 saddr` la IPv6, y en Python agrupar una lista mixta genera un `TypeError` porque ambas versiones no pueden combinarse. Una lista de prefijos IPv4 no dice nada sobre los clientes IPv6, que pasan a la siguiente regla o comportamiento. Mantenga prefijos IPv4 e IPv6 en paralelo para cada regla y compruebe ambos al probar.

Lecturas adicionales

- RFC 4632: enrutamiento entre dominios sin clases (CIDR), el plan de asignación y agregación de direcciones de internet — la descripción actual de CIDR, con su historia, tabla de prefijos y reglas de agregación.
- RFC 3021: uso de prefijos de 31 bits en enlaces IPv4 punto a punto — por qué un /30 gasta la mitad de sus direcciones en un enlace punto a punto y cómo lo evita un /31.
- RFC 4291: arquitectura de direccionamiento de IP versión 6 — las reglas de direccionamiento IPv6, incluidos los prefijos con la misma notación de barra.
- ipaddress: biblioteca de manipulación de IPv4/IPv6 — la referencia de Python para analizar, dividir, comparar y agrupar prefijos.
- nft(8): manual de nftables — la referencia de nftables, incluido el orden de las reglas y qué veredictos terminan una cadena.
- wg(8): establecer y obtener la configuración de interfaces WireGuard — el manual de WireGuard, donde AllowedIPs se define como una lista de direcciones con máscaras CIDR.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es CIDR?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/cidr/