Advanced MetalLB configuration

June 12, 2023 | Federico Paolinelli | 8-minute read

Hybrid cloud

< Back to all posts

MetalLB is a load-balancer implementation for bare-metal Kubernetes clusters that uses standard routing protocols. It is widely used to allow external access to applications running on the cluster, and its adoption ranges from homelab users to big telecommunications service providers.

It is actively developed by its upstream community and supported by Red Hat in the OpenShift Container Platform.

Based on user feedback, the API of MetalLB evolved to provide new options and different semantics compared to the original, configmap-based version.

In this post, I will explore the MetalLB API and how to leverage it to address complex networking topologies, but first, I'll refresh how MetalLB works and the problems it tries to solve.

A service of type LoadBalancer requires two things:

- A virtual IP that can be used to reach the service
- A way to advertise the IP address, so that the traffic directed to it can reach the nodes

Once the traffic gets to the nodes, the container network interface of the cluster (OVN Kubernetes or OpenShift-SDN in OpenShift) is responsible for driving the traffic to the endpoints.

Note: this is not meant to replace the official upstream or OpenShift documentation by any means, but to show how the various configuration degrees can be combined to implement complex scenarios.

Address allocation

MetalLB needs to be provided with the list of IP addresses that can be assigned to the LoadBalancer services.

The entity responsible for this is IPAddressPool:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: example
  namespace: metallb-system
spec:
  addresses:
  - 192.168.10.0/24
  - 192.168.9.1-192.168.9.5
  - fc00:f853:0ccd:e799::/124
```

The IP ranges can be created with the CIDR notation or the “from-to” notation. An IPAddressPool can contain multiple ranges, and both IPV4 and IPV6 ranges. You can create multiple IPAddressPools.

Note that in the case of dual stack services, MetalLB will select both IPs from the same IPAddressPool. When an IPAddressPool is created, the IP gets assigned to the service, but it is not advertised. In order to advertise it, a BGPAdvertisement or an L2Advertisement must be created.

Allocation mechanics

After a service is created, MetalLB will choose an available IP from the nonallocated IPs available from all the configured IPAddressPools (excluding those with autoassign=false) and assign it to the service. When the service is deleted, MetalLB restores the IP to the pool it belongs to so it can be reused for a new service.

By default, the IP allocation is arbitrary. However, it can be influenced in multiple ways.

The service is pinned to a specific IP

This can be done by specifying the IP via the field spec.loadBalancerIP or by using the custom metallb.universe.tf/loadBalancerIPs annotation. In the case of dual stack services, only the annotation can be used.

The service is pinned to a specific IPAddressPool

This can be done by specifying the pool you want the IP from via the metallb.universe.tf/address-pool annotation.

Using the IPAddressPool allocation

This is the configuration option to use if the cluster administrator wants to control the allocation of the various pools across different applications or tenants, avoiding the risk of a workload to pick the wrong IP. The IPAddressPool allocation can be driven by the namespace of the service, or by a service selector, or by both. In this way, the cluster administrator is in control of how the IP allocation is associated with the various services.

In the following example, the IPAddressPool is allocatable only to services belonging to namespaces matching the given selector:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: ippool-ns-service-alloc-sample
  namespace: metallb-system
spec:
  addresses:
    - 192.168.20.0/24
  serviceAllocation:
    priority: 50
    namespaceSelectors:
      - matchLabels:
          app: myapp
```

The namespaces the pool is allocated to can be specified as a list of namespaces:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: ippool-ns-service-alloc-sample
```

```
    namespace: metallb-system
  spec:
    addresses:
      - 192.168.20.0/24
    serviceAllocation:
      priority: 50
      namespaces:
        - namespace-a
        - namespace-b
```

On top of that, it is possible to refine the granularity at the service level:

```
  apiVersion: metallb.io/v1beta1
  kind: IPAddressPool
  metadata:
    name: ippool-ns-service-alloc-sample
    namespace: metallb-system
  spec:
    addresses:
      - 192.168.20.0/24
    avoidBuggyIPs: true
    serviceAllocation:
      priority: 50
      namespaces:
        - namespace-a
      serviceSelectors:
        - matchExpressions:
            - {key: app, operator: In, values: [bar]}
```

Given a Service, MetalLB will sort the matching IPAddressPool based on priority and use the first with available IPs. In case multiple IPAddressPools have the same priority, the choice is nondeterministic.

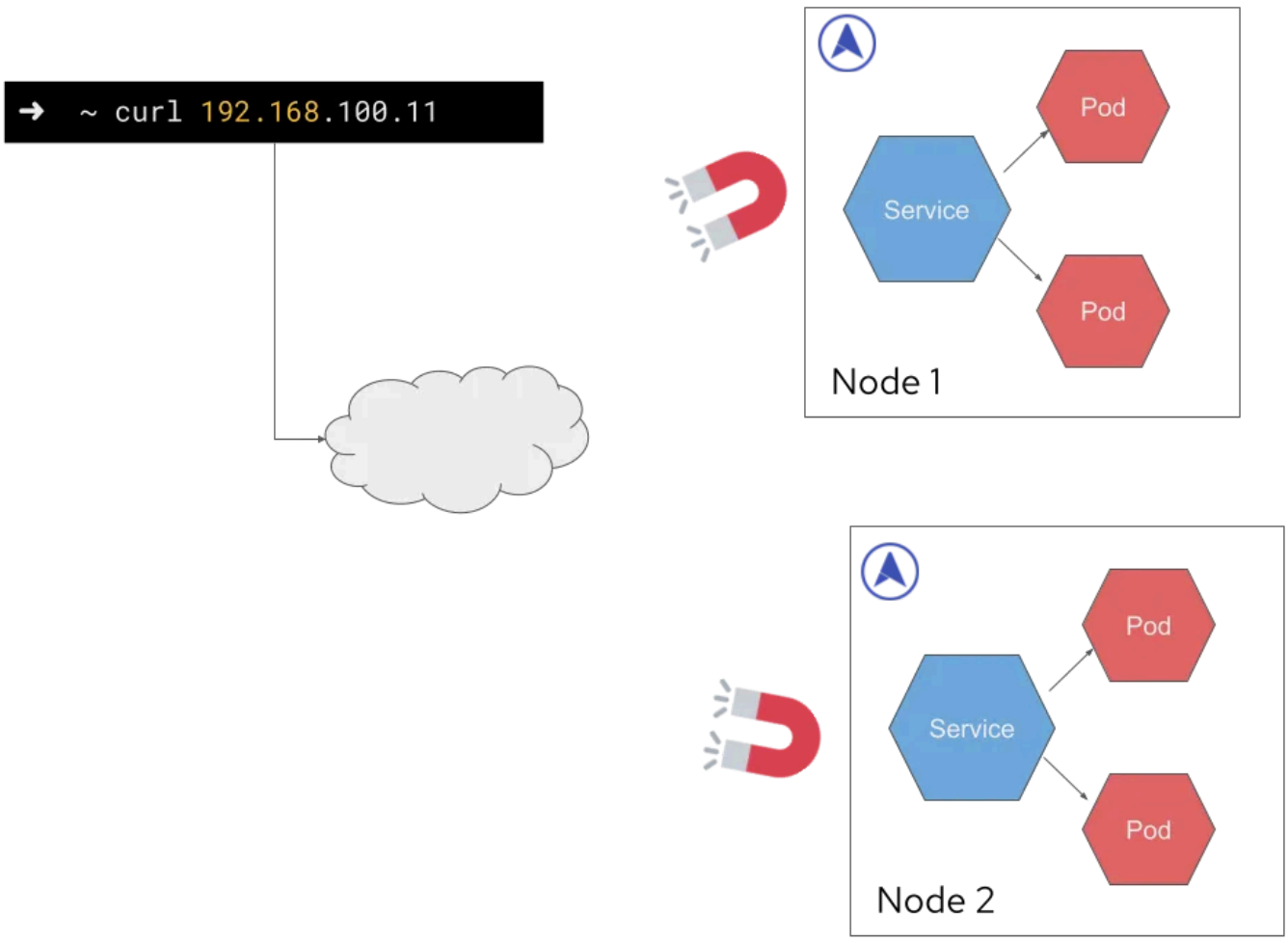
Notes:

- No selector means the IPAddressPool can be applied to all the services.
- A lower value of the priority means the priority is higher, but no priority is the lowest.

- If a service specifies an IP or a pool, but the corresponding IPAddressPool selector doesn't match the service, the IP won't be allocated to the service.
- Currently, if a configuration change would cause a service to receive a different IP (i.e., selector change, priority change), the service's IP will be preserved to avoid traffic disruptions. The service must be recreated in order to have it fetch the new IP.

Advertising the services

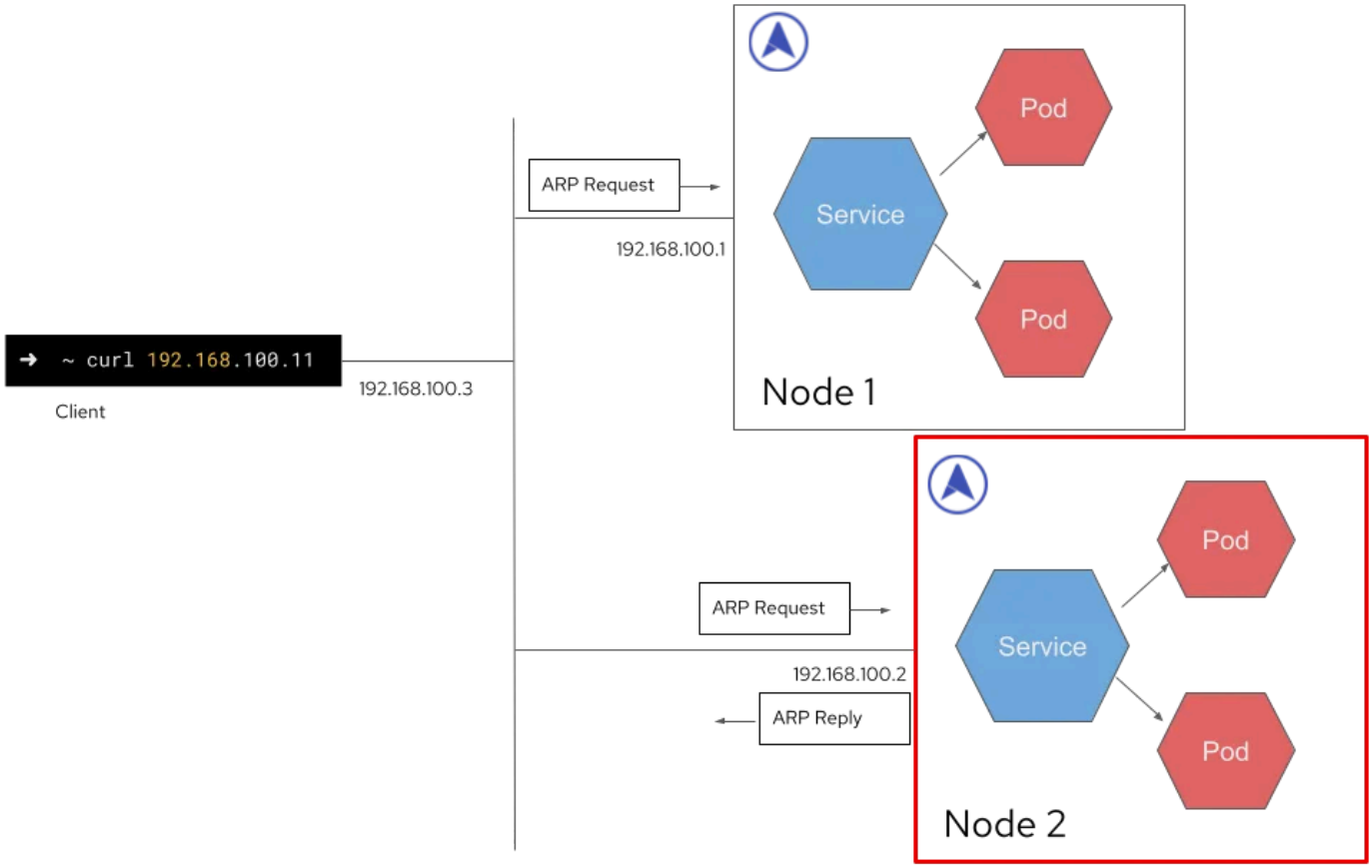
The second problem MetalLB solves is making traffic directed to the LoadBalancer IP reach the cluster nodes.



MetalLB currently offers two different ways to attract the traffic directed to the service's IP to a given node. I won't describe the differences in how the two implementations work here, as it is covered already in the upstream documentation.

Advertise via L2 mode

In L2 mode, MetalLB behaves similarly to Keepalived. It replies to ARP requests asking for the LoadBalancerIP with the MAC address of the interface it received the request from.



To announce a service via L2, you must create an L2Advertisement instance. The simple version of an L2Advertisement does not have any fields:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: empty
  namespace: metallb-system
```

An empty L2Advertisement will enable MetalLB to advertise all the services belonging to all the pools via L2.

By providing a pool selector, you can limit the services announced via this L2Advertisement only to those whose IP is provided by one of the selected pools. In the following example, only the services with IPs coming from the first-pool IPAddressPool will be advertised.

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
```

```
    name: example
    namespace: metallb-system
  spec:
    ipAddressPools:
    - first-pool
```

Limit the nodes advertising the service via L2

When working in L2 mode, MetalLB will advertise the service from one of the nodes eligible for announcing the service. By using a node selector, you can restrict the set of nodes that can be elected to announce the service:

```
  apiVersion: metallb.io/v1beta1
  kind: L2Advertisement
  metadata:
    name: example
    namespace: metallb-system
  spec:
    nodeSelectors:
    - matchLabels:
        network: controlplane
```

When announcing a service using L2 mode, only one node is elected as the entry point. Hence, this configuration option is particularly important when only a subset of the nodes are connected to the network you want to advertise the service to.

Limit the interfaces to reply from

If you use the interfaces selector in the L2Advertisement, MetalLB will reply to ARP requests only if they come from one of the selected interfaces.

```
  apiVersion: metallb.io/v1beta1
  kind: L2Advertisement
  metadata:
    name: example
    namespace: metallb-system
  spec:
    ipAddressPools:
```

```
    - third-pool
    interfaces:
    - eth0
    - eth1
```

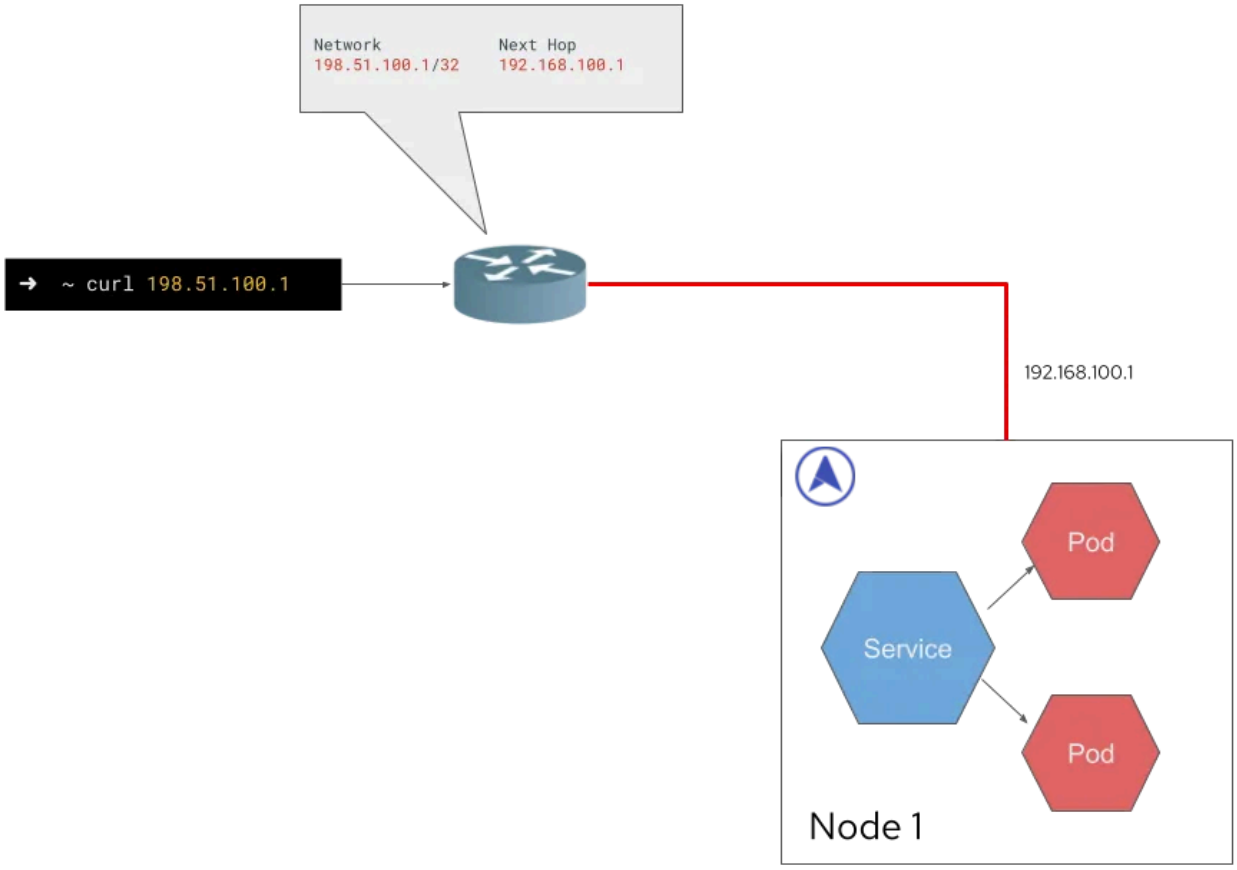
This approach is useful when multiple interfaces are connected to the same subnet, or with complex interfaces such as bridges or Linux switches.

Notes:

- The interface selector does not influence the way the leader for a given IP is selected. If the node does not have any of those interfaces, the service won't be announced.
- When the IPAddressPool selector is not specified, the L2 Advertisement is applied to all the IPAddressPools.
- When multiple Advertisements are applied to the same IPAddressPool, they are combined and applied together. This means that the eligible nodes are the union of all the nodes selected by the IPAddressPools, and the interfaces are the union of the selected interfaces.

Advertise via BGP mode

When operating in BGP mode, MetalLB acts as a router running on every node of the cluster. It establishes a BGP session with any configured router and announces a route to the LoadBalancerIP going through all the nodes of the cluster. In this way, the traffic is balanced across the nodes by ECMP routes.



The BGPPeer CRD must be used to instruct MetalLB on how to establish a BGP session with an external router.

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: peer
  namespace: metallb-system
spec:
  myASN: 64512
  peerASN: 64512
  peerAddress: 172.30.0.3
```

In order to advertise the services via BGP, a BGPAdvertisement must be created. The simple version of a BGPAdvertisement does not have any fields:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
```

```
  metadata:
    name: empty
    namespace: metallb-system
```

By doing this, all the services with IPs belonging to all the IPAddressPools will be announced via BGP to all the configured BGPPeers.

As with L2Advertisement, a pool selector is available to limit the services affected by a given BGPAdvertisement:

```
  apiVersion: metallb.io/v1beta1
  kind: BGPAdvertisement
  metadata:
    name: example
    namespace: metallb-system
  spec:
    ipAddressPools:
    - first-pool
```

In BGP, there are three degrees of freedom that can be combined:

- Which nodes to announce the service from
- Which peers to announce the service to
- Which peers to establish a BGP session to

Announcing the service from a subset of the nodes

A node selector is available to limit the nodes advertised as the next hop for the service.

```
  apiVersion: metallb.io/v1beta1
  kind: BGPAdvertisement
  metadata:
    name: advertisement
    namespace: metallb-system
  spec:
    aggregationLength: 32
    aggregationLengthV6: 128
    ipAddressPools:
```

```
    - first-pool
    nodeSelectors:
    - matchLabels:
        role: serviceAdvertiser
```

By doing this, MetalLB will advertise only the nodes matching the given labels as next hops.

Announcing the service only to a subset of peers

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: example
  namespace: metallb-system
spec:
  ipAddressPools:
  - PoolA
  peers:
  - PeerA
  - PeerB
```

By specifying a peer selector, MetalLB will advertise the service only to the BGPPeers specified in the BGPAdvertisement. This allows you to announce the services with IPs provided by PoolA only over specific BGP sessions.

Establishing the BGP session from a subset of nodes

If a router is not reachable from a subset of the nodes, there is no point in attempting to connect. Because of this, it is possible to limit the nodes you establish BGP sessions from using a node selector on the BGPPeer:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: example
  namespace: metallb-system
spec:
```

```
    myASN: 64512
    peerASN: 64512
    peerAddress: 172.30.0.3
    peerPort: 180
    nodeSelectors:
    - matchLabels:
        rack: frontend
```

Notes:

- No IPAddressPool selector means that the given BGPAdvertisement is applied to all the services.
- As per the L2Advertisement, when multiple BGPAdvertisements are applied to the same IPAddressPool, they are combined and applied together.

Putting everything together

MetalLB is widely used, both in its community form and as part of the OpenShift platform. Its user base ranges from homelab users to big organizations with complex topologies and requirements.

Its configuration is meant to be simple in order to accommodate basic use cases while also allowing complex configurations achieved by combining all the configuration options listed here.

For example, we can express the following configurations:

- Announcing both via L2 and via BGP
- Announcing via L2, but only on the nodes with a given label and only from interface eth2
- Announcing to a given BGP neighbor, but only from the nodes with a given label
- Announcing all the services to all the BGP neighbors, but only some services to a subset of the BGP neighbors
- Announcing the services belonging to a namespace only via a specific BGP session

Additionally, combining the IPAddressPool namespace/service selector and the IPAddressPool selector in the advertisements allows crafting an announcement behavior specific to a particular workload.

ABOUT THE AUTHOR

Federico Paolinelli

More like this

BLOG POST

Cloud-native at your pace: Why the guide you choose matters

BLOG POST

Introducing IdM in RHEL domain join feature: Enroll your machines on boot!

ORIGINAL SHOWS

Edge computing covered and diced | Technically Speaking

ORIGINAL SHOWS

Crack the Cloud_Open | Command Line Heroes

Browse by channel

Explore all channels →

Automation

The latest on IT automation for tech, teams, and environments

→

Artificial intelligence

Updates on the platforms that free customers to run AI workloads anywhere

→

Open hybrid cloud

Explore how we build a more flexible future with hybrid cloud

→

Security

The latest on how we reduce risks across environments and technologies

Edge computing

Updates on the platforms that simplify operations at the edge

Infrastructure

The latest on the world's leading enterprise Linux platform

Applications

Inside our solutions to the toughest application challenges

Original shows

Entertaining stories from the makers and leaders in enterprise tech




Products

Tools

Try, buy, & sell

Communicate

About Red Hat

We're the world's leading provider of enterprise open source solutions—including Linux, cloud, container, and Kubernetes. We deliver hardened solutions that make it easier for enterprises to work across platforms and environments, from the core datacenter to the network edge.

Select a language

English