FRR-k8s as a BGP backend for MetalLB

July 5, 2024 | Federico Paolinelli, Ori Braunshtein | 5-minute read

Hybrid cloud Network automation Containers

< Back to all posts

The FRR-k8s API provides important Border Gateway Protocol (BGP) features required by the MetalLB load balancer with the ability to selectively accept routes and to inject raw FRR configuration options. It's built on FRRouting (FRR), a free and open source internet routing protocol suite for Linux and Unix platforms that implements Border Gateway Protocol (BGP) and Bidirectional Forwarding Detection (BFD).

Why you need FRR-k8s

MetalLB is a Kubernetes load balancer implementation that allows exposing Kubernetes services through BGP. The MetalLB project has replaced its native golang BGP implementation of MetalLB with FRR, and we've found that users want to leverage FRR for purposes beyond what MetalLB could do. For instance, a user might want to use a single FRR instance to receive routes on Kubernetes nodes through BGP. We've provided a short term solution, but we're also thinking about a cleaner way to leverage the same FRR instance to serve both MetalLB and other actors (such as users or another Kubernetes controller).

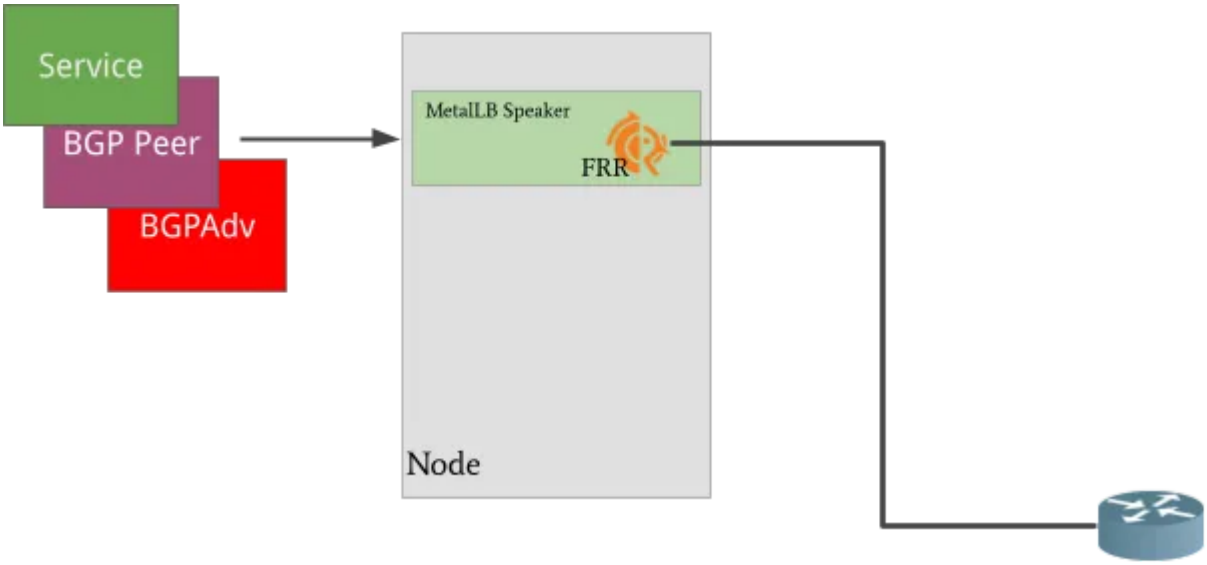
These goals have converged into a design proposal discussed and accepted by the community. The idea is to have an instance of FRR external to MetalLB, available through an API using

CustomResourceDefinition (CRD), that's consumable from MetalLB but also available directly to the user. The design proposal has evolved in the implementation of a new project: FRR-k8s.

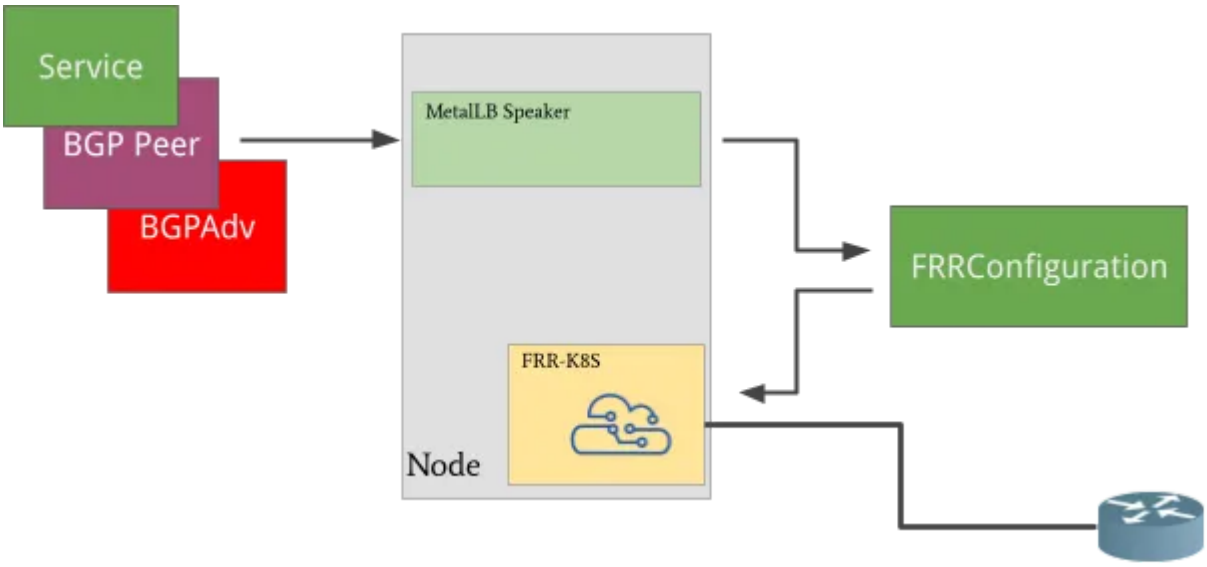
The community version of FRR-K8s is meant to be deployable as a standalone component, or it can be deployed together with MetalLB if its new FRR-K8s backend option is chosen, which is the main subject of this blog post.

The architecture

Currently, FRR runs as a sidecar container of the MetalLB speaker pod, which reconciles events from Kubernetes to the proper FRR configuration for the FRR instance running inside the pod.



On the other hand, when the BGP backend is implemented with FRR-K8s, MetalLB generates the proper configuration for FRR-K8s. This in turn translates that configuration into a valid configuration for its sidecar FRR container, which in turn implements the BGP protocol. For example:



Introduction to the FRR-K8s API

The FRRConfiguration CRD is used to configure an FRR instance. For example, the following configuration advertises a single prefix to a given neighbor, and allows all incoming prefixes to be accepted:

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
 name: example
 namespace: FRR-k8s-system
spec:
 bgp:
   routers:
   - asn: 64512
     neighbors:
     - address: 172.18.0.5
       asn: 64512
       toAdvertise:
         allowed:
           prefixes:
           - 192.168.100.0/24
       toReceive:
         allowed:
           mode: all
     prefixes:
     - 192.168.100.0/24
     - 192.169.101.0/24
```

We won't cover all the details of the FRRConfiguration structure here, but here are the important points in this config:

- configures a BGP session with a given neighbor
- declares what prefixes are advertised for each neighbor
- declares what prefixes are accepted from a given neighbor

The FRRConfiguration CRD also comes with a node selector field that allows you to specify a configuration for a subset of nodes in a cluster:

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
name: sample-node-selector
namespace: FRR-k8s-system
spec:
```

```
  bgp:
    routers:
    - asn: 64512
      neighbors:
      - address: 172.30.0.3
        asn: 64512
  nodeSelector:
    labelSelector:
      foo: "bar"
```

Experimental raw FRR configuration

It's still experimental, but the API also allows you to add a snippet of raw FRR configuration that's added after the one rendered by the daemon. The `priority` field determines what order multiple raw configurations are appended in. This is an experimental feature, so **do not use it in production!**

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
name: sample-raw
namespace: FRR-k8s-system
spec:
raw:
  priority: 5
  rawConfig: |-
    router bgp 64512
      neighbor 172.18.0.5 remote-as 4200000000
      neighbor 172.18.0.5 timers 0 0
      neighbor 172.18.0.5 port 179
```

The FRRNodeState resource

The FRR-K8s daemon generates an FRRNodeState instance for each node it’s running on.

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRNodeState
metadata:
 name: FRR-k8s-worker
spec: {}
status:
```

```
  lastConversionResult: success
  lastReloadResult: success
  runningConfig: |
    Building configuration...
    Current configuration:
    !
    frr version 8.4.2_git
    frr defaults traditional
    hostname FRR-k8s-worker
    log file /etc/frr/frr.log informational
    !
    router bgp 64512
     neighbor 172.18.0.5 remote-as 64512
```

The status lists:

- result of processing all the FRRConfigurations applicable to a given node
- result of applying the configuration to the FRR instance
- configuration that the FRR instance is currently running with

Merging multiple configurations together

So far, we've talked about the option to have multiple FRRConfigurations for a node, but we haven't explained how it works. This is how multiple FRRConfigurations can be merged together. When it comes to having multiple FRRConfigurations for a node, the guiding principles are:

- A configuration must be self-contained (it cannot rely on others)
- A configuration can only add to the existing state
- A more permissive configuration can override a less permissive one

It's possible to add a new neighbor to a router, or to advertise an additional prefix to a neighbor. However, there is no way to remove a component added by another configuration (for example, you can't remove a neighbor specified in another FRRConfiguration).

In some cases, there could be conflicting configurations that can't be applied together. For example:

- different ASN for the same router (in the same VRF)
- different ASN for the same neighbor (with the same IP and port)
- multiple BFD profiles with the same name, but different values

When the daemon cannot merge a set of configurations for a node, it reports the configuration as invalid (through metrics and the Status resource) and leaves the previous valid configuration in

place.

Integration with MetalLB

When running Red Hat OpenShift in the new FRR-K8s mode, MetalLB does not directly configure the FRR instance by itself. Instead, it creates and manages the relevant FRRConfiguration, which in turn is handled by the FRR-K8s daemon.

Specifically, each MetalLB speaker consumes its K8s objects (BGPPeers, LoadBalancer Services, and so on) as usual, and generates a single FRRConfiguration from them. That FRRConfiguration is named `metallb-<NODE-NAME>` and has a selector to match only the node with that particular name. This ensures that a node consumes the MetalLB configuration coming only from its own speaker.

For example, suppose that a MetalLB speaker on a node named `blognode` has a BGPPeer:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
 name: blogpeer
 namespace: metallb-system
spec:
 myASN: 64512
 peerASN: 64512
 peerAddress: 172.18.0.5
```

A BGPAdvertisement advertises a LoadBalancer service with IP 192.168.10.100 to that peer. In this case, the MetalLB speaker would create the following FRRConfiguration:

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
 name: metallb-blognode
 namespace: metallb-system
spec:
 bgp:
   routers:
   - asn: 64512
     neighbors:
     - address: 172.18.0.5
       asn: 64512
       toAdvertise:
         allowed:
```

```
        mode: filtered
        prefixes:
        - 192.168.10.100/32
    prefixes:
    - 192.168.10.100/32
```

The speaker continuously manages this FRRConfiguration, reacting to any changes to relevant Kubernetes objects, updating it accordingly.

How to use FRR-K8s in Red Hat OpenShift

As of OpenShift 4.16, the field **bgpBackend** has been added to the MetalLB Operator's **MetalLB** CRD. By default, MetalLB is deployed in the usual **FRR** mode. To deploy it in **FRR-K8s** mode, set **bgpBackend** to `FRR-k8s` . Once it's set, the Operator deploys the FRR-k8s daemon on the cluster, and MetalLB consumes its API.

Conclusion

In this post, we've discussed FRR-k8s, its architecture, how it can be used to allow multiple actors to configure an FRR daemon on an OpenShift node, and how MetalLB can leverage it. FRR-k8s allows a node to act as an FRR BGP router in a Kubernetes-compliant manner. The API is fully documented in the README file of the community project's Git repo, and in the documentation for OpenShift 4.16.

ABOUT THE AUTHORS

Federico Paolinelli

Ori Braunshtein

More like this

BLOG POST

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

BLOG POST

The time is right for telcos to break free

ORIGINAL SHOWS

Get into GitOps | Technically Speaking

ORIGINAL SHOWS

Kubernetes and the quest for a control plane | Technically Speaking

Keep exploring

Building Resilient Microservices with Istio and Red Hat OpenShift Service Mesh
Training

Network automation guide E-book

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