8 months ago
· kubernetes (/tag/kubernetes/), ipv6 (/tag/ipv6/), network (/tag/network/), calico (/tag/calico/), docker (/tag/docker/), cluster (/tag/cluster/)
· 9 Comments

Kubernetes in IPv6 only (/kubernetes-ipv6-only/)

Since few weeks, I'm working to deploy IPv6-only Kubernetes cluster in production for my company. Here is how I made it, maybe it will help you if you want to try this (tumultuous) adventure.

I- Kubernetes installation

To put Kubernetes in production, I strongly recommend avoiding using automated installation script but instead, use a configuration management tool (Saltstack, Chef, Puppet...) to deploy Kubernetes from scratch like done in **kubernetes-the-hard-way** by the awesome Kelsey Hightower (https://github.com/kelseyhightower) (the k8s-dope guy).

This installation way, with a configuration manager, ensures that you always got your binaries and configuration present and consistent on your Kubernetes servers (master/worker/etcd) and make easy Kubernetes updates and rollback if needed.

In my case I deploy Kubernetes 1.8.x on Debian 9.x KVM virtual-machines with IPv6 only networking on a dedicated VLAN configured by Saltstack. I use Ferm (IPtables front-end) on each VMs to ensure strictly defined firewall rules (to restrain ETCD access for example).

Each Kubernetes components are IPv6 capable, so you can connect your API to your ETCD and then your Kubelets to your API and finally talk to your API with kubectl without any IPv4.

II- C.N.I configuration

You need to choose a CNI provider (Container Network Interface) to handle your Containers networking in Kubernetes. My choice was directly done for **Project Calico** which provide pure Layer 3 networking (Linux IP Routes) with BGP routes exchange between workers (pretty awesome).

Calico allows you easily to enable IPv6 and also to totally disable IPv4 on your pod's side. Kubernetes networking with Calico is configured in the CNI configuration file (`/etc/cni/net.d/` by default) and come with the **calicoctl** tool that makes possible to create new IPPools in a running cluster. **Calico-node** run on every workers and need an ETCD access to store his states (can use the same as k8s).

By default Calico use a Unique Local Address (https://en.wikipedia.org/wiki/Unique_local_address) (ULA) IPv6 range. If you want to make quick tests with it, you need to allow NAT to make sure that your pod can talk to public IPv6 networks. To do that you need to delete the IPPool and then create the same (or another) with NAT enabled :

```
calicoctl delete ippool fd80:24e2:f998:72d6::/64

cat <<EOF | calicoctl create -f -
- apiVersion: v1
  kind: ipPool
  metadata:
    cidr: fd80:24e2:f998:72d6::/64
  spec:
    nat-outgoing: true
EOF
```

In production way, you will not use a ULA IPv6 pool because in the version 6 of the IP protocol everything need to be public and without NAT (it's not a good practice to use NAT in IPv6).

To use public IPv6 networking for your pods, you need to configure a Calico's BGP Peer to announce to your company router your Pod's IPv6 routes handled by Kubernetes workers, it's done quite easily with a `.yaml` file like this :

```
apiVersion: v1
kind: bgpPeer
metadata:
  peerIP: 2001:DB8:f::100
  scope: global
spec:
  asNumber: 65538
```

Calico breaks by default IPv6 pool into /**122 by workers** and randomly takes addresses from these pools according to the worker on which the pod is scheduled.

Another great feature of Calico is that you can indicate which IPPool you want to use in a pod Deployment definition with a simple annotation, for example :

```
annotations:
      "cni.projectcalico.org/ipv6pools": "[\"2001:db8::1/120\"]
```

More references here (https://docs.projectcalico.org/v2.2/reference/cni-plugin/configuration).

III- Services and Kube-DNS

One of the biggest problems when using IPv6 cluster is about ClusterIP (**10.32.0.0/16** by default). You need to totally stop using them, by declaring `clusterIP: None` on every defined Service. With that, you can speak to your Service like `my-database.default.svc.cluster-domain.tld` which will be resolved with Service's IPv6 Endpoints instead of Service's ClusterIP.

With Kubernetes in IPv6 only, Kube-Proxy become totally useless.

Another small problem is that some Kubernetes applications try directly to talk to your API via **10.32.0.1** which for sure is unreachable.
To solve this problem you need to pass theses environment variables to your deployments :

```
    - name: KUBERNETES_SERVICE_HOST
      value: public-api.domain.tld
    - name: KUBERNETES_SERVICE_PORT
      value: '6443'
```

If you plan to deploy Kubernetes with IPv6 in production, any good way to achieve a scalable kube-dns architecture, is to deploy a couple of DNS deployment (4 for example) using well-known defined IPv6 address (with Calico annotations) that allow you to get 4 non-changing public IPv6 that you can use in your DNS records as a NS delegation, to make your cluster domain resolution public.

With that, you're able to use public DNS (like IPv6 Google one (https://developers.google.com/speed/public-dns/docs/using)) in your Pods, and when you will try to speak to an internal domain name (`my-app.default.svc.cluster-domain.tld`) the DNS answers will be given by any of your Kube-DNS pods.

IV- Expose services to users

To expose my External Kubernetes services in IPv6-only I use the Nginx ingress-controller, that use Endpoint as virtual-host back end, so it works from scratch on IPv6 with latest image :

```
gcr.io/google_containers/nginx-ingress-controller:0.9.0-beta.15
```

I deploy a couple of Nginx-ingress in the cluster with well-known IPv6 that we can directly use in our DNS records to expose Services publicly. You can also use **kube-lego** to automatically fetch Let's Encrypt certificates.

If you want to expose external services in IPv4/6, you will need to setup an External LoadBalancer (like done in AWS or GCP) with a couple of High-Available Nginx or HAProxy with dual-stack networking pointing on this IPv6 only pods. It's what we do in our cluster.

V- Persistent Volumes

We mostly use GlusterFS as a persistent volume provider.

We were surprised to see that Gluster work in IPv6-only without any special tweaking. You need to configure it to bind on the wanted IP address, then you can peer your servers to create a Gluster cluster.

After that, you can easily create Persistent Volume on your Kubernetes cluster that point to external IPv6-only GlusterFS cluster.

To achieve this point you need to simply create a endpoint/service for your external Gluster cluster.

VI- Pods security

In this topology, all the pods can talk to each other. It's like a big swimming-pool for all your pods. It's not the best for security.

With Calico Policy Controller, you can define simple Network Policy (k8s) rules that say things like `nobody can talk to this namespace except pods with the label "my-frontend"`. It's like migrating from big swimming pool

to a small private trusted jacuzzis.

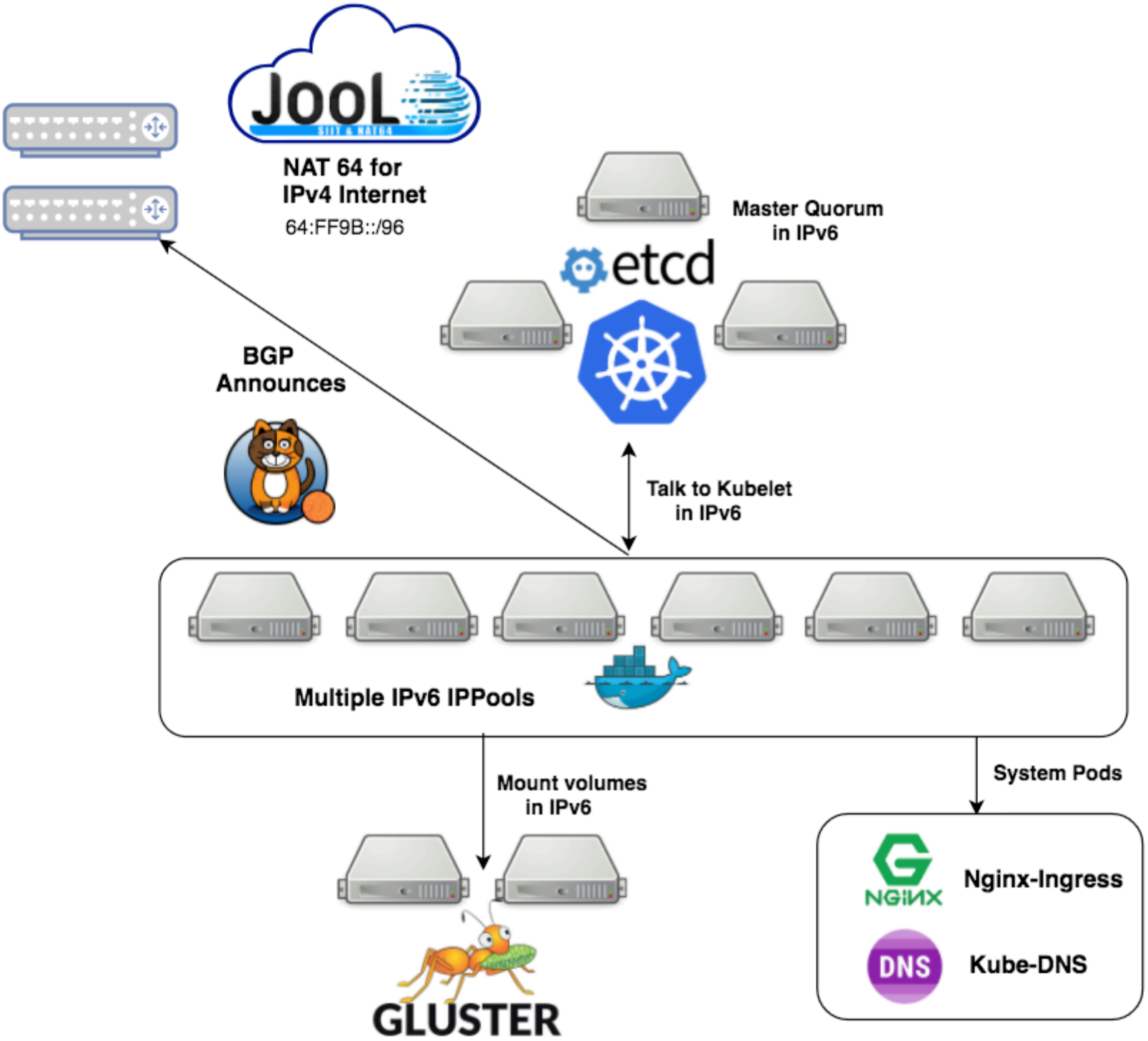
The Policy controller work in IPv6-only from scratch using IP6tables.

VII- Conclusion

Deploying a production-wide IPv6-only Kubernetes cluster was a big challenge for us but a required step because we are working in IPv6-only VM network so it was impossible to keep IPv4 long time only for our containers without trying IPv6-only.

As you should know, IPv6 is not IPv4 compatible so to be able to reach IPv4 world from our version 6 network, we use Jool (http://jool.mx/en/index.html) to enable NAT64 at our DNS side to be able to join IPv4-only websites like the famous Github.com.

One last point that make IPv6-only not user-friendly is because, before try new services from Internet in our cluster, we need to make few changes to make it work like `clusterIP: None` Services and put the right environment variables if pods need to talk to k8s API. Otherwise, it's just pleasure.



Let me know if you got any question !

I would like to thank especially Tigera (https://tigera.io/) (the company behind Project Calico (https://www.projectcalico.org/)) for help provided and for the investigations on CNI problems I encountered. Thanks !

WRITTEN BY
Valentin Ouvrard (https://valentin.ouvrard.it)
127.0.0.1
https://valentin.ouvrard.it
(https://valentin.ouvrard.it)

Published on
November 14, 2017

SPREAD THE WORD