Nate Baker Follow
Oct 4, 2017 · 5 min read

HA Kubernetes with Kubeadm

Updated for 1.9

This post is geared towards users who are already using Kubeadm to deploy their Kubernetes clusters. If you're not familiar with Kubeadm, check it out here: https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/

An HA guide has been added to the official Kubernetes documentation: https://kubernetes.io/docs/setup/independent/high-availability/

Background

I'm creating this in hopes that it helps people with their Kubernetes journey. When first starting out, I followed the very popular "Kubernetes The Hardway"guide: https://github.com/kelseyhightower/kubernetes-the-hard-way. This is a great starting point for people who want to understand the ins and outs of how Kubernetes runs. If you're building from scratch and running on bare metal, VMware, AWS, GCP, or wherever—the process is going to be similar. While its good to know how all of the Kubernetes pieces connect, it can be time consuming.

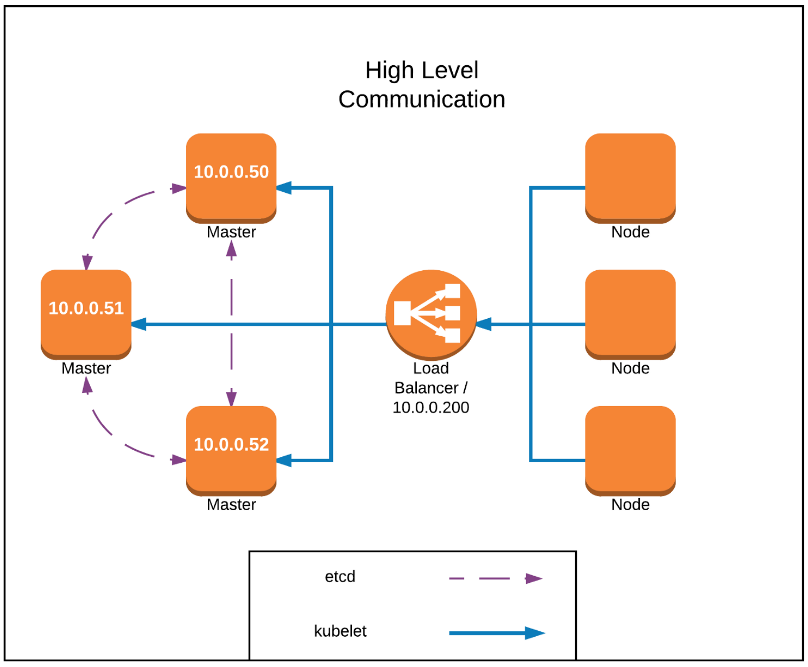
Enter Kubeadm

I've since changed the way I'm building and deploying clusters by leveraging Kubeadm. This tool is great for standing up a k8s cluster. However it has one glaring limitation—it does not support multi-master environments. This is something that is being actively worked on, but for now you have to handle high availability on your own.

I'm not going to go into detail about standing up a single node master cluster. The official docs are great.

Creating a multi-master cluster

When creating a multi-master cluster, there is an order of operations that you need to follow. I'm going to assume that you've already setup your servers with the necessary prerequisites. Below is a high level diagram of what we're trying to achieve. Feel free to make adjustments where you need to.



We're using a Load Balancer for our Node => Master communication. You could achieve the same setup on AWS with an elastic load balancer, or by leveraging HAproxy. You just need something to load balance traffic for the master nodes.

Let's assume the following:

```
master 1 address      = 10.0.0.50
master 2 address      = 10.0.0.51
master 3 address      = 10.0.0.52
load balancer address = 10.0.0.200
```

For this to work, you will need an external etcd cluster. I'm assuming that you've already created a functional cluster on three master nodes. Make sure etcd and your masters are setup before you continue!

Step 1—Prep your instances

Enter the kubeadm config file.

Before we deploy the config file, we need to generate a token.

`kubeadm token generate`

Save this output, we're going to use it in our config file.

Kubeadm config file:

```
apiVersion: kubeadm.k8s.io/v1alpha1
kind: MasterConfiguration
etcd:
  endpoints:
  - "http://10.0.0.50:2379"
  - "http://10.0.0.51:2379"
  - "http://10.0.0.52:2379"
apiServerCertSANs:
- "10.0.0.50"
- "10.0.0.51"
- "10.0.0.52"
- "10.0.0.200"
- "127.0.0.1"
token: "YOUR KUBEADM TOKEN"
tokenTTL: "0"
```

In the config file, we need to tell kubeadm to use the external etcd cluster. If you setup etcd for tls communication, include those details. Also include the additional SANs for our API Server certificate. Make sure to include all master addresses *AND* the load balancer address!

Now distribute this config to master instances.

Step 2—Kubeadm init

Once the config is distributed, we can start initializing the first master.

```
# on the first master instance:
kubeadm init --config /path/to/config.yaml
```

On a successful initialization, we'll have a single master that is ready to roll. The new master instance will have all the certificates and keys necessary for our master cluster.

For Kubernetes 1.9 make sure you copy the `discovery-token-ca-cert-hash` . You will need this when joining worker nodes.

You will see output similar to:

```
kubeadm join --token $YOUR_KUBE_TOKEN 10.0.0.50:6443 --
discovery-token-ca-cert-hash
sha256:89870e4215b92262c5093b3f4f6d57be8580c3442ed6c8b00b0b3
0822c41e5b3
```

Don't forget to setup the home directory with the kubectl admin.conf file, and apply the network overlay! *(I use weave)*

Step 3—Distribute the PKI

Assuming that the first master was created successfully, we should have a directory with all of our certs and keys in `/etc/kubernetes/pki` . This directory will need to be distributed to the other 2 master instances.

Copy the `pki` directory to `/etc/kubernetes` on the other masters. I'll leave it up to you on how to distribute it.

Step 4—Initialize the other masters

Now that our certs and keys are setup, its time to initialize the other master servers.

Run `kubeadm init --config /path/to/config.yaml`

You'll notice the output is slightly different—kubeadm will detect that the `/etc/kubernetes/pki` directory already exists and is populated with certificates. It skips the pki creation, assuming that the certs and keys are valid and there for a reason.

Step 5—Setup kubectl

If the above steps are complete, and your load balancer is setup to handle the masters, then we should have a highly available master setup! Alright!

Lets verify our cluster is working by setting up `kubectl` to communicate with our load balancer.

Setup `kubectl` like you would normally by copying `/etc/kubernetes/admin.conf` into `~/.kube/config`

Edit `~/.kube/config` and change `server: 10.0.0.50` to `server: 10.0.0.200`

Run `kubectl get nodes` and you should see output similar to:

```
NAME          STATUS    ROLES     AGE       VERSION
10.0.0.50     Ready     master    1h        v1.9.0
10.0.0.51     Ready     master    1h        v1.9.0
10.0.0.52     Ready     master    1h        v1.9.0
```

Step 6—Join nodes to the cluster

Now that we have a functional master cluster, lets join some worker nodes!

On any of your workers, run:

```
kubeadm join --token YOUR_CLUSTER_TOKEN 10.0.0.200:6443
```

For Kubernetes 1.9 include your `discovery-token-ca-cert-hash`

```
kubeadm join --token YOUR_CLUSTER_TOKEN 10.0.0.200:6443 --
discovery-token-ca-cert-hash
sha256:89870e4215b92262c5093b3f4f6d57be8580c3442ed6c8b00b0b3
0822c41e5b3
```

And thats it! If everything was setup cleanly, you should now have a highly available cluster. Start joining workers, and take over the world!

Closing Thoughts

Using the kubeadm config file is still in alpha, things *will* inevitably change as kubeadm becomes more mature. If you're running on a cloud provider, like AWS, you can add those things into the config file. You can still automate all of this, it just takes a little TLC.

Hopefully the process of distributing PKI and leveraging a load balancer becomes more streamlined.

This is just one way to achieve HA with kubeadm. I probably missed a few things. Please feel free to make adjustments, and leave suggestions in the comments!

Thanks!

Nate