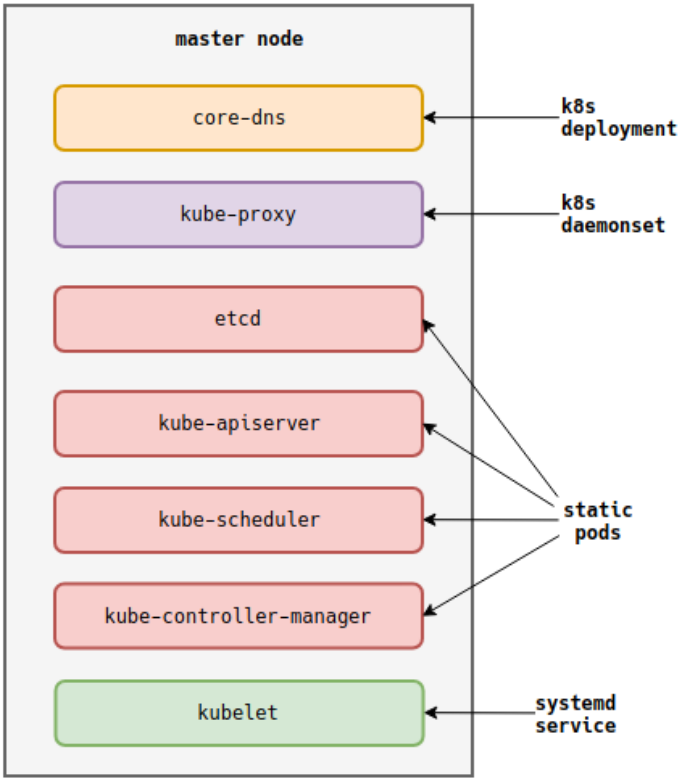


> This represents the default. Through additional configuration, `kubeadm init` can behave differently, such as reusing an existing etcd cluster.

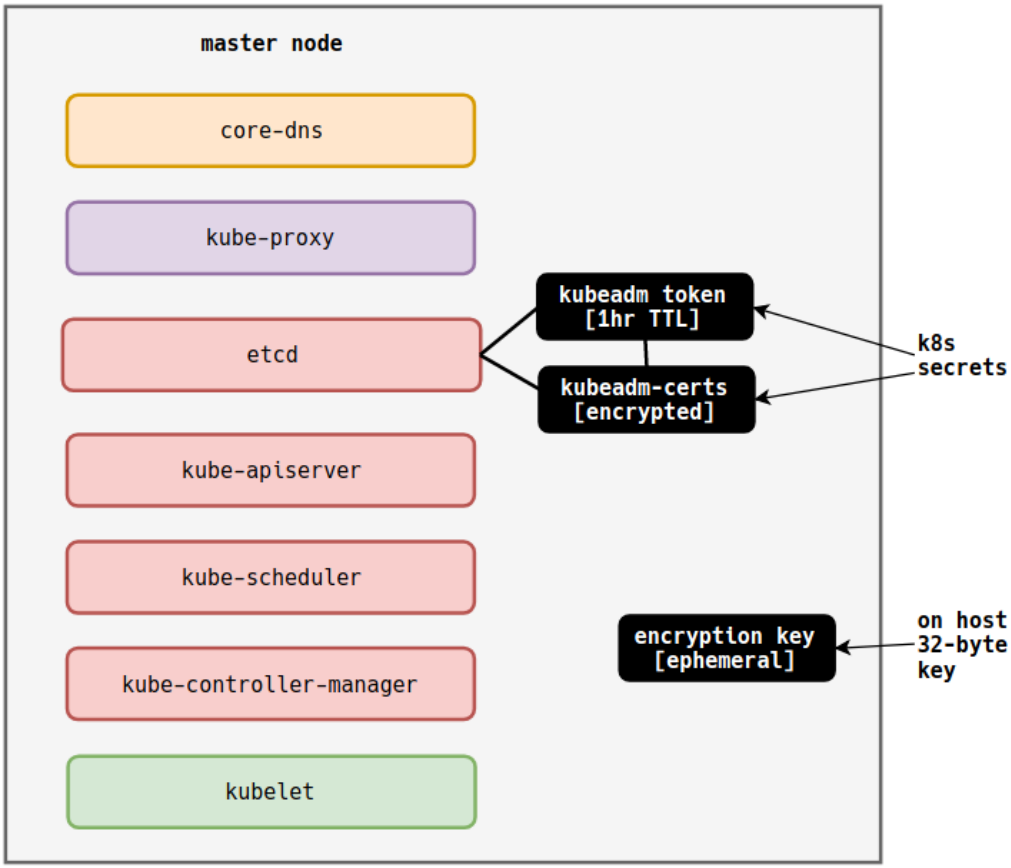
Conventionally, after installing a CNI plugin, users copy PKI information across 2 more master nodes and run a `kubeadm` command to add new control plane nodes. This results in a 3 node control plane.

1.14 introduced the `--upload-certs` flag to the `kubeadm init` command. As detailed in the KEP, it has the following impact.

1. Creates an encryption key on the host.
2. Encrypts certificates and keys with the encryption key.
3. Adds encrypted data to the `kubeadm-certs` secret in the `kube-system` namespace.
4. Links the `kubeadm-certs` secret to a kubeadm token with a **1 hour TTL**.

Taking the example above, if we run the following command, it will result in the new diagram below.

```
kubeadm init --experimental-upload-certs
```



When the kubeadm token expires, so does the `kubeadm-certs` secret. Also, whenever the `init phase upload-certs` is run, a new encryption key is created. Ensuring that if `kube-apiserver` is compromised during the adding of a master node, the secret is encrypted and meaningless to the attacker. This example demonstrates running `--experimental-upload-certs` during cluster bootstrap. It is also possible to tap into phases, using `kubeadm init phase upload-certs` to achieve the above on an existing master. This will be detailed in the walkthrough below.

Lastly, it is important to note the token generated during `upload-certs` is only a proxy used to bind a TTL to the `kubeadm-cert` secret. You still need a conventional `kubeadm` token to join a control plane. This is the same token you would need to join a worker.

To add another control plane (master) node, a user can run the following command.

```
kubeadm join ${API_SERVER_PROXY_IP}:${API_SERVER_PROXY_PORT} \
    --experimental-control-plane \
    --certificate-key=${ENCRYPTION_KEY} \
    --token ${KUBEADM_TOKEN} \
    --discovery-token-ca-cert-hash ${APISERVER_CA_CERT_HASH}
```

Walkthrough: Creating the HA Control Plane

This walkthrough will guide you to creating a new Kubernetes cluster with a 3 node control plane. It will demonstrate joining a control plane right after bootstrap and how to add another control plane node after the bootstrap tokens have expired.

1. Create 4 hosts (vms, baremetal, etc).

   These hosts will be referred to as `loadbalancer`, `master0`, `master1`, and `master2`.

   master hosts may need 2 vCPUs and 2GB of RAM available. You can get around these requirements during testing by ignoring pre-

flight checks. See the kubeadm documentation for more details.

2. Record the host IPs for later use.

Setup the Load Balancer

In this section, we'll run a simple NGINX load balancer to provide a single endpoint for our control plane. This load balancer example is not meant for production scenarios.

1. SSH to the `loadbalancer` host.
2. Create the directory `/etc/nginx`.

```
mkdir /etc/nginx
```

3. Add and edit the file `/etc/nginx/nginx.conf`.

```
vim /etc/nginx/nginx.conf
```

4. Inside the file, add the following configuration.

```
events { }

stream {
    upstream stream_backend {
        least_conn;
        # REPLACE WITH master0 IP
        server 192.168.122.160:6443;
        # REPLACE WITH master1 IP
        server 192.168.122.161:6443;
        # REPLACE WITH master2 IP
        server 192.168.122.162:6443;
    }

    server {
        listen        6443;
        proxy_pass    stream_backend;
        proxy_timeout 3s;
        proxy_connect_timeout 1s;
    }

}
```

5. Alter each line above with a `REPLACE` comment above it.
6. Start NGINX.

```
docker run --name proxy \
    -v /etc/nginx/nginx.conf:/etc/nginx/nginx.conf:ro \
    -p 6443:6443 \
    -d nginx
```

7. Verify you can reach NGINX at its address.

```
curl 192.168.122.170
```

output:

```
curl: (52) Empty reply from server
```

Install Kubernetes Binaries on Master Hosts

1. Complete all pre-requisites and installation on master nodes.
    a. See: https://kubernetes.io/docs/setup/independent/install-kubeadm.

Initialize the Cluster and Examine Certificates

1. SSH to the `master0` host.
2. Create the directory `/etc/kubernetes/kubeadm`

```
mkdir /etc/kubernetes/kubeadm
```

3. Create and edit the file `/etc/kubernetes/kubeadm/kubeadm-config.yaml`.

```
vim /etc/kubernetes/kubeadm/kubeadm-config.yaml
```

4. Add the following configuration.

```
apiVersion: kubeadm.k8s.io/v1beta1
kind: ClusterConfiguration
kubernetesVersion: stable
# REPLACE with `loadbalancer` IP
controlPlaneEndpoint: "192.168.122.170:6443"
networking:
  podSubnet: 192.168.0.0/18
```

5. Alter the line with a REPLACE comment above it.
6. Initialize the cluster with `upload-certs` and `config` specified.

```
kubeadm init \
    --config=/etc/kubernetes/kubeadm/kubeadm-config.yaml \
    --experimental-upload-certs
```

7. Record the output regarding joining control plane nodes for later use.

   output:

```
You can now join any number of the control-plane node running the
following command on each as root:

kubeadm join 192.168.122.170:6443 --token nmiqmn.yls76lcyxg2wt36c \
--discovery-token-ca-cert-hash
sha256:5efac16c86e5f2ed6b20c6dbcbf3a9daa5bf75aa604097dbf49fdc3d1fd5ff7d \
--experimental-control-plane --certificate-key
828fc83b950fca2c3bda129bcd0a4ffcd202cfb1a30b36abb901de1a3626a9df
```

   Note the `certificate-key` that enables decrypting the kubeadm certs secret.

8. As your user, run the recommended kubeconfig commands for `kubectl` access.

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

9. Examine the `kubeadm-cert` secret in the `kube-system` namespace.

```
kubectl get secrets -n kube-system kubeadm-certs -o yaml
```

You should see certs for etcd, kube-apiserver, and service accounts.

10. Under `ownerReferences`, examine the `name`.

```
name: bootstrap-token-cwb9ra
```

This correlates to an ephemeral kubeadm token. When that token expires, so does this secret.

11. List available tokens with `kubeadm`.

```
kubeadm token list
```

output:

```
TOKEN                     TTL       EXPIRES                USAGES
DESCRIPTION
cwb9ra.gegoj2eqddaf3yps   1h        2019-03-26T19:38:18Z   <none>
Proxy for managing TTL for the kubeadm-certs secret
nmiqmn.yls76lcyxg2wt36c   23h       2019-03-27T17:38:18Z
authentication,signing   <none>
```

Note that `cwb9ra` is the owner reference in the above step. This is **not** a join token, instead a proxy that enables ttl on `kubeadm-certs`. We still need to use the `nmiqmn` token when joining.

12. Install `calico` CNI-plugin with a pod CIDR matching the `podSubnet` configured above.

```
kubectl apply -f
https://gist.githubusercontent.com/joshrosso/ed1f5ea5a2f47d86f536e9eee3f1a
2c2/raw/dfd95b9230fb3f75543706f3a95989964f36b154/calico-3.5.yaml
```

13. Verify 1 node is `Ready`.

```
kubectl get nodes
```

output:

```
NAME              STATUS   ROLES    AGE   VERSION
192-168-122-160   Ready    master   79m   v1.14.0
```

14. Verify `kube-system` pods are `Running`.

```
kubectl get pods -n kube-system
```

output:

```
NAME                                    READY   STATUS    RESTARTS   AGE
calico-node-mphpw                       1/1     Running   0          58m
coredns-fb8b8dccf-c6s9q                 1/1     Running   0          80m
coredns-fb8b8dccf-mxzrm                 1/1     Running   0          80m
etcd-192-168-122-160                    1/1     Running   0          79m
kube-apiserver-192-168-122-160          1/1     Running   0          79m
```

```
kube-controller-manager-192-168-122-160   1/1     Running   0          79m
kube-proxy-dpxhx                          1/1     Running   0          80m
kube-scheduler-192-168-122-160            1/1     Running   0          79m
```

Add the Second Master

1. SSH to the `master1` host.
2. Run the recorded join command from the previous section.

```
kubeadm join 192.168.122.170:6443 --token nmiqmn.yls76lcyxg2wt36c \
--discovery-token-ca-cert-hash
sha256:5efac16c86e5f2ed6b20c6dbcbf3a9daa5bf75aa604097dbf49fdc3d1fd5ff7d \
--experimental-control-plane \
--certificate-key
828fc83b950fca2c3bda129bcd0a4ffcd202cfb1a30b36abb901de1a3626a9df
```

3. After completion, verify there are now 2 nodes.

```
kubectl get nodes
```

   output:

```
NAME              STATUS   ROLES    AGE   VERSION
192-168-122-160   Ready    master   22m   v1.14.0
192-168-122-161   Ready    master   34s   v1.14.0
```

4. Verify new pods have been created.

```
kubectl get pods -n kube-system
```

   output:

```
NAME                                      READY   STATUS    RESTARTS   AGE
calico-node-cq5nt                         1/1     Running   0          60s
calico-node-spn5w                         1/1     Running   0          13m
coredns-fb8b8dccf-r9sc8                   1/1     Running   0          22m
coredns-fb8b8dccf-wlcm4                   1/1     Running   0          22m
etcd-192-168-122-160                      1/1     Running   0          21m
etcd-192-168-122-161                      1/1     Running   0          59s
kube-apiserver-192-168-122-160            1/1     Running   0          21m
kube-apiserver-192-168-122-161            1/1     Running   0          59s
kube-controller-manager-192-168-122-160   1/1     Running   0          21m
kube-controller-manager-192-168-122-161   1/1     Running   0          60s
kube-proxy-tflhf                          1/1     Running   0          60s
kube-proxy-vthjr                          1/1     Running   0          22m
kube-scheduler-192-168-122-160            1/1     Running   0          22m
kube-scheduler-192-168-122-161            1/1     Running   0          59s
```

Add the Third Master with New Tokens

This section joins the third and final master. However, we will first delete all existing `kubeadm` tokens. This approach demonstrates how you could add masters when the Kubernetes cluster is already running.

1. On an existing master, list all tokens.

```
kubeadm token list
```

2. Delete all existing tokens.

```
kubeadm token delete cwb9ra.gegoj2eqddaf3yps
kubeadm token delete nmiqmn.yls76lcyxg2wt36c
```

> Now the previously recorded join command will not work as the kubeadm-certs secret has expired and been deleted, the encryption key is no longer valid, and the join token will not work.

3. Create a new token with a 10 minute TTL.

```
kubeadm token create --ttl 10m --print-join-command
```

output:

```
kubeadm join 192.168.122.170:6443 \
    --token xaw58o.0fjg0xp0ohpucwhr \
    --discovery-token-ca-cert-hash
sha256:5efac16c86e5f2ed6b20c6dbcbf3a9daa5bf75aa604097dbf49fdc3d1fd5ff7d
```

> Note the IP above must reflect your `loadbalancer` host.

4. Run the `upload-certs` phase of `kubeadm init`.

```
kubeadm init phase upload-certs --experimental-upload-certs
```

output:

```
[upload-certs] Storing the certificates in ConfigMap "kubeadm-certs" in
the "kube-system" Namespace
[upload-certs] Using certificate key:
9555b74008f24687eb964bd90a164ecb5760a89481d9c55a77c129b7db438168
```

5. SSH to the `master2` host.

6. Use the outputs from the previous steps to run a control-plane join command.

```
kubeadm join 192.168.122.170:6443 \
    --experimental-control-plane \
    --certificate-key
9555b74008f24687eb964bd90a164ecb5760a89481d9c55a77c129b7db438168 \
    --token xaw58o.0fjg0xp0ohpucwhr \
    --discovery-token-ca-cert-hash
sha256:5efac16c86e5f2ed6b20c6dbcbf3a9daa5bf75aa604097dbf49fdc3d1fd5ff7d
```

7. After completion, verify there are now 3 nodes.

```
kubectl get nodes
```

output:

```
NAME              STATUS   ROLES    AGE     VERSION
192-168-122-160   Ready    master   50m     v1.14.0
192-168-122-161   Ready    master   28m     v1.14.0
192-168-122-162   Ready    master   3m30s   v1.14.0
```

Summary