Cloud Provider:AWS
Server:Ubuntu 18.04 x6
Server List: 2 Controller(Master), 2(Worker), 1 Nginx(Reverse Proxy), 1 Ubuntu 18.04(local)

What is the Kubernetes Control Plane?

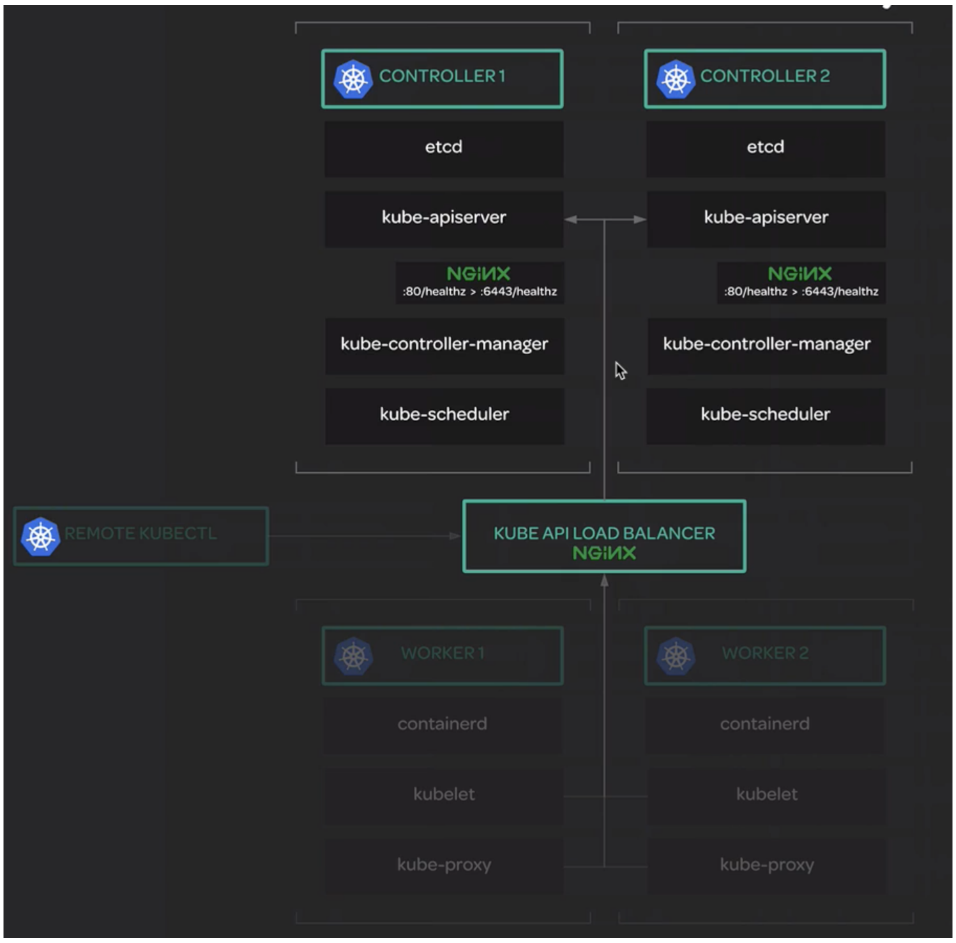
- The Kubernetes control plane is a set of services that control the Kubernetes cluster.
- Control plane components “make global decisions about the cluster(e.g.,scheduling) and detect and respond to cluster events(e.g.,starting up a new pod when a replication controller’s replicas field is unsatisfied).”

Kubernetes Control Plane Components

- kube-apiserver: Serves the Kubernetes API. This allows users to interact with the cluster.
- etcd: Kubernetes cluster datastore.
- kube-scheduler: Schedules pods on available worker nodes.
- kube-controller-manager: Runs a series of controllers that provides a wide range of functionality.
- cloud-controller-manager: Handles interaction with underlying cloud providers.

// The control plane components will need to be installed on each controller node

Control Plane Architecture Overview



Installing Kubernetes Control Plane Binaries

// You can install the control plane binaries on each control node like this:

```
$ sudo mkdir -p /etc/kubernetes/config
```

```
$ wget -q --show-progress --https-only --timestamping \
"https://storage.googleapis.com/kubernetes-release/release/v1.10.2/bin/linux/amd64/kube-apiserver" \
"https://storage.googleapis.com/kubernetes-release/release/v1.10.2/bin/linux/amd64/kube-controller-manager" \
"https://storage.googleapis.com/kubernetes-release/release/v1.10.2/bin/linux/amd64/kube-scheduler" \
"https://storage.googleapis.com/kubernetes-release/release/v1.10.2/bin/linux/amd64/kubectl"
```

```
$ chmod +x kube-apiserver kube-controller-manager kube-scheduler kubectl
```

```
$ sudo mv kube-apiserver kube-controller-manager kube-scheduler kubectl /usr/local/bin/
```

// The Kubernetes API server provides the primary interface for the Kubernetes control plane and the cluster as a whole. When you interact with Kubernetes, you are nearly always doing it through the Kubernetes API server. This lesson will guide you through the process of configuring the kube-apiserver service on your two Kubernetes control nodes. After completing this lesson, you should have a systemd unit set up to run kube-apiserver as a service on each Kubernetes control node.

// You can configure the Kubernetes API server like so:

```
$ sudo mkdir -p /var/lib/kubernetes/
```

```
$ sudo cp ca.pem ca-key.pem kubernetes-key.pem kubernetes.pem \
  service-account-key.pem service-account.pem \
  encryption-config.yaml /var/lib/kubernetes/
```

// Set some environment variables that will be used to create the systemd unit file. Make sure you replace the placeholders with their actual values:

```
$ INTERNAL_IP=$(curl http://169.254.169.254/latest/meta-data/local-ipv4)
```

```
$ CONTROLLER0_IP=<private ip of controller 0>
```

```
$ CONTROLLER1_IP=<private ip of controller 1>
```

// Generate the kube-apiserver unit file for systemd :

```
$ cat << EOF | sudo tee /etc/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
```

```
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-apiserver \\
  --advertise-address=${INTERNAL_IP} \\
  --allow-privileged=true \\
  --apiserver-count=3 \\
  --audit-log-maxage=30 \\
  --audit-log-maxbackup=3 \\
  --audit-log-maxsize=100 \\
  --audit-log-path=/var/log/audit.log \\
  --authorization-mode=Node,RBAC \\
  --bind-address=0.0.0.0 \\
  --client-ca-file=/var/lib/kubernetes/ca.pem \\

--enable-admission-plugins=NamespaceLifecycle,NodeRestriction,LimitRanger,ServiceAcco
unt,DefaultStorageClass,ResourceQuota \\
  --enable-swagger-ui=true \\
  --etcd-cafile=/var/lib/kubernetes/ca.pem \\
  --etcd-certfile=/var/lib/kubernetes/kubernetes.pem \\
  --etcd-keyfile=/var/lib/kubernetes/kubernetes-key.pem \\
  --etcd-servers=https://$CONTROLLER0_IP:2379,https://$CONTROLLER1_IP:2379 \\
  --event-ttl=1h \\
  --experimental-encryption-provider-config=/var/lib/kubernetes/encryption-config.yaml \\
  --kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \\
  --kubelet-client-certificate=/var/lib/kubernetes/kubernetes.pem \\
  --kubelet-client-key=/var/lib/kubernetes/kubernetes-key.pem \\
  --kubelet-https=true \\
  --runtime-config=api/all \\
  --service-account-key-file=/var/lib/kubernetes/service-account.pem \\
  --service-cluster-ip-range=10.32.0.0/24 \\
  --service-node-port-range=30000-32767 \\
  --tls-cert-file=/var/lib/kubernetes/kubernetes.pem \\
  --tls-private-key-file=/var/lib/kubernetes/kubernetes-key.pem \\
  --v=2 \\

--kubelet-preferred-address-types=InternalIP,InternalDNS,Hostname,ExternalIP,ExternalDNS
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
```

Setting up the Kubernetes Controller Manager

//Now that we have set up kube-apiserver , we are ready to configure kube-controller-manager. This lesson walks you through the process of configuring a systemd service for the Kubernetes Controller Manager. After completing this lesson, you should have the kubeconfig and systemd unit file set up and ready to run the kube-controller-manager service on both of your control nodes.

// You can configure the Kubernetes Controller Manager like so:

```
$ sudo cp kube-controller-manager.kubeconfig /var/lib/kubernetes/
```

// Generate the kube-controller-manager systemd unit file:

```
$ cat << EOF | sudo tee /etc/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-controller-manager \\
  --address=0.0.0.0 \\
  --cluster-cidr=10.200.0.0/16 \\
  --cluster-name=kubernetes \\
  --cluster-signing-cert-file=/var/lib/kubernetes/ca.pem \\
  --cluster-signing-key-file=/var/lib/kubernetes/ca-key.pem \\
  --kubeconfig=/var/lib/kubernetes/kube-controller-manager.kubeconfig \\
  --leader-elect=true \\
  --root-ca-file=/var/lib/kubernetes/ca.pem \\
  --service-account-private-key-file=/var/lib/kubernetes/service-account-key.pem \\
  --service-cluster-ip-range=10.32.0.0/24 \\
  --use-service-account-credentials=true \\
  --v=2
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
```

Setting up the Kubernetes Scheduler

// Now we are ready to set up the Kubernetes scheduler. This lesson will walk you through the process of configuring the kube-scheduler systemd service. Since this is the last of the three control plane services that need to be set up in this section, this lesson also guides you through enabling and starting all three services on both control nodes. Finally, this lesson shows you how to verify that your Kubernetes controllers are healthy and working so far. After completing this lesson, you will have a basic, working, Kubernetes control plane distributed across your two control nodes.

// You can configure the Kubernetes Scheduler like this.
// Copy kube-scheduler.kubeconfig into the proper location:

```
$ sudo cp kube-scheduler.kubeconfig /var/lib/kubernetes/
```

// Generate the kube-scheduler yaml config file.

```
$ cat << EOF | sudo tee /etc/kubernetes/config/kube-scheduler.yaml
apiVersion: componentconfig/v1alpha1
kind: KubeSchedulerConfiguration
clientConnection:
  kubeconfig: "/var/lib/kubernetes/kube-scheduler.kubeconfig"
leaderElection:
  leaderElect: true
EOF
```

// Create the kube-scheduler systemd unit file:

```
$ cat << EOF | sudo tee /etc/systemd/system/kube-scheduler.service
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-scheduler \\
  --config=/etc/kubernetes/config/kube-scheduler.yaml \\
  --v=2
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
```

// Start and enable all of the services:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable kube-apiserver kube-controller-manager kube-scheduler
$ sudo systemctl start kube-apiserver kube-controller-manager kube-scheduler
```

// It's a good idea to verify that everything is working correctly so far:
// Make sure all the services are active (running) :

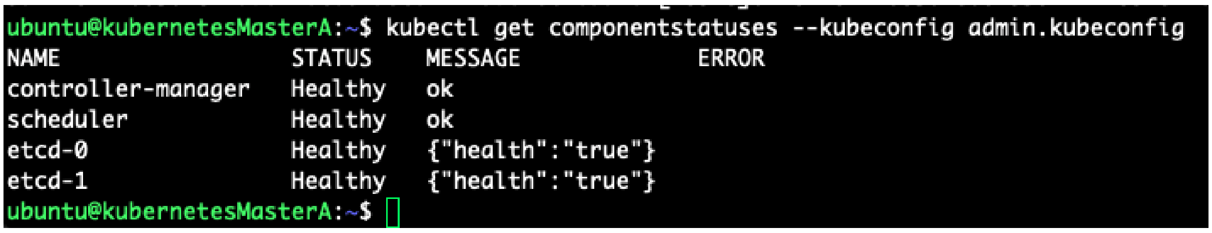
```
$ sudo systemctl status kube-apiserver kube-controller-manager kube-scheduler
```

// Use kubectl to check componentstatuses:

$ kubectl get componentstatuses --kubeconfig admin.kubeconfig

// You should get output that looks like this:

```
NAME STATUS MESSAGE ERROR
controller-manager Healthy ok
scheduler Healthy ok
etcd-0 Healthy {"health": "true"}
etcd-1 Healthy {"health": "true"}
```



```
ubuntu@kubernetesMasterA:~$ kubectl get componentstatuses --kubeconfig admin.kubeconfig
NAME                 STATUS    MESSAGE             ERROR
controller-manager   Healthy   ok
scheduler            Healthy   ok
etcd-0               Healthy   {"health":"true"}
etcd-1               Healthy   {"health":"true"}
ubuntu@kubernetesMasterA:~$
```

Enable HTTP Health Checks

Why do we need to enable HTTP health checks?

- The load balancer needs to be able to perform health checks against the Kubernetes API to measure the health status of API nodes.

// Part of Kelsey Hightower's original Kubernetes the Hard Way guide involves setting up an nginx proxy on each controller to provide access to the Kubernetes API /healthz endpoint over http. This lesson explains the reasoning behind the inclusion of that step and guides you through the process of implementing the http /healthz proxy.

```
$ sudo curl -k https://localhost:6443/healthz          // does work
$ sudo curl -k http://localhost:6443/healthz           // won't work
```

// You can set up a basic nginx proxy for the healthz endpoint by first installing nginx":

```
$ sudo apt-get install -y nginx
```

// Create an nginx configuration for the health check proxy:

```
$ cat > kubernetes.default.svc.cluster.local << EOF
server {
  listen      80;
  server_name kubernetes.default.svc.cluster.local;

  location /healthz {
    proxy_pass                    https://127.0.0.1:6443/healthz;
    proxy_ssl_trusted_certificate /var/lib/kubernetes/ca.pem;
  }
}
EOF
```

// Set up the proxy configuration so that it is loaded by nginx:

```
$ sudo mv kubernetes.default.svc.cluster.local
/etc/nginx/sites-available/kubernetes.default.svc.cluster.local
$ sudo ln -s /etc/nginx/sites-available/kubernetes.default.svc.cluster.local
/etc/nginx/sites-enabled/
$ sudo systemctl restart nginx
$ sudo systemctl enable nginx
```

// You can verify that everything is working like so:

```
$ curl -H "Host: kubernetes.default.svc.cluster.local" -i http://127.0.0.1/healthz
```

// You should receive a 200 OK response.

Set up RBAC for Kubelet Authorization

Why do we need to set up RBAC for Kubelet Authorization

- Must make sure that the Kubernetes API has permission to access the Kubelet API on each node and perform certain common tasks. Without this, some functionality will not work.
- We will create a ClusterRole with necessary permissions and assign that role to the Kubernetes user with a ClusterRoleBinding.

// One of the necessary steps in setting up a new Kubernetes cluster from scratch is to assign permissions that allow the Kubernetes API to access various functionality within the worker kubelets. This lesson guides you through the process of creating a ClusterRole and binding it to the kubernetes user so that those permissions will be in place. After completing this lesson, your cluster will have the necessary role-based access control configuration to allow the cluster's API to access kubelet functionality such as logs and metrics.

// You can configure RBAC for kubelet authorization with these commands. Note that these commands only need to be run on one control node.

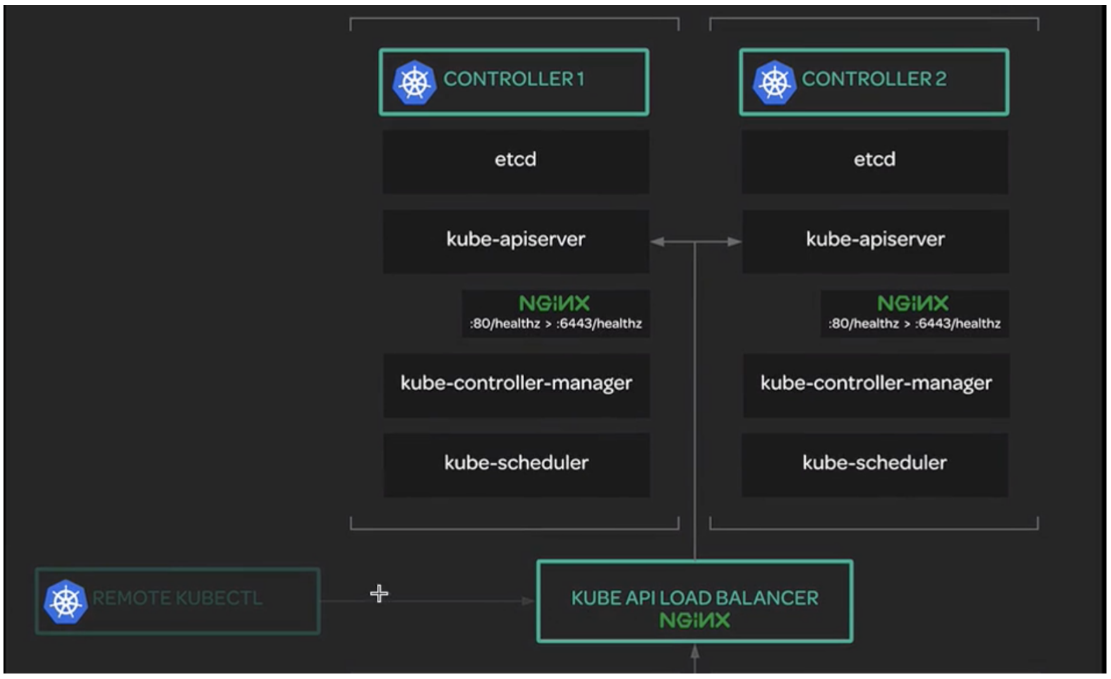
// Create a role with the necessary permissions:

```
$ cat << EOF | kubectl apply --kubeconfig admin.kubeconfig -f -
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:kube-apiserver-to-kubelet
rules:
  - apiGroups:
      - ""
    resources:
      - nodes/proxy
      - nodes/stats
      - nodes/log
      - nodes/spec
      - nodes/metrics
    verbs:
      - "*"
EOF
```

// Bind the role to the kubernetes user:

```
$ cat << EOF | kubectl apply --kubeconfig admin.kubeconfig -f -
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: system:kube-apiserver
  namespace: ""
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:kube-apiserver-to-kubelet
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: User
    name: kubernetes
EOF
```

Setting up a Kube API Frontend Load Balancer



// In order to achieve redundancy for your Kubernetes cluster, you will need to load balance usage of the Kubernetes API across multiple control nodes. In this lesson, you will learn how to create a simple nginx server to perform this balancing. After completing this lesson, you will be able to interact with both control nodes of your kubernetes cluster using the nginx load balancer.

// Here are the commands you can use to set up the nginx load balancer. Run these on the server that you have designated as your load balancer server:

```
$ sudo apt-get install -y nginx

$ sudo systemctl enable nginx

$ sudo mkdir -p /etc/nginx/tcpconf.d

$ sudo vi /etc/nginx/nginx.conf
```

// Add the following to the end of nginx.conf:

```
include /etc/nginx/tcpconf.d/*;
```

// Set up some environment variables for the lead balancer config file:

```
$ CONTROLLER0_IP=<controller 0 private ip>
$ CONTROLLER1_IP=<controller 1 private ip>
```

// Create the load balancer nginx config file:

```
$ cat << EOF | sudo tee /etc/nginx/tcpconf.d/kubernetes.conf
stream {
    upstream kubernetes {
        server $CONTROLLER0_IP:6443;
        server $CONTROLLER1_IP:6443;
    }

    server {
        listen 6443;
        listen 443;
        proxy_pass kubernetes;
    }
}
EOF
```

// Reload the nginx configuration:

```
$ sudo nginx -s reload
```

// You can verify that the load balancer is working like so:

```
$ curl -k https://localhost:6443/version
```