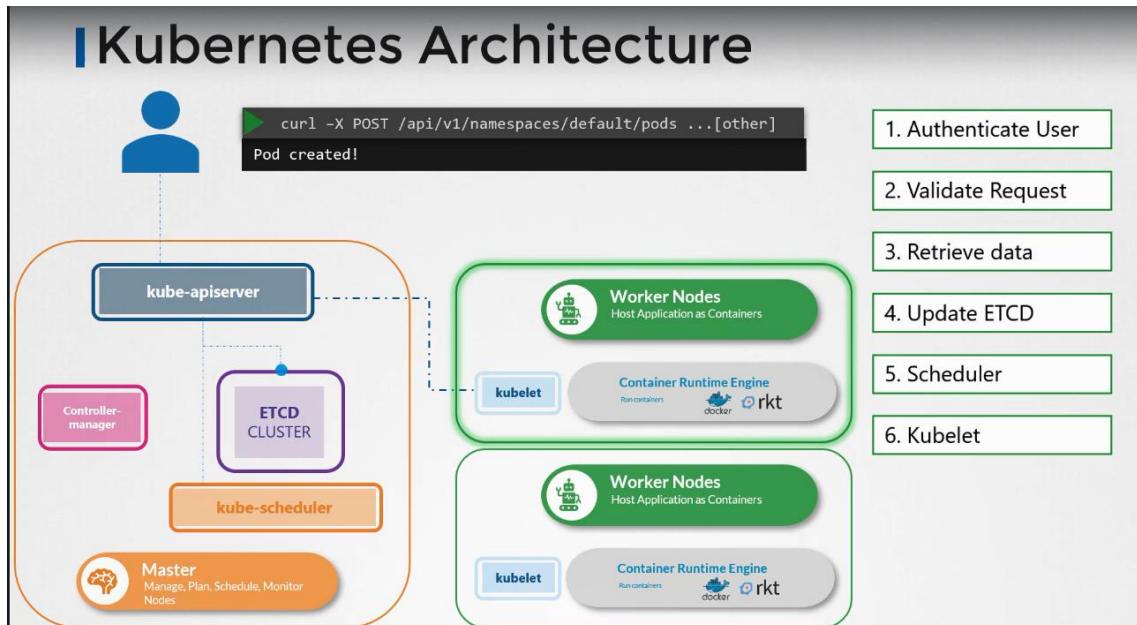


CKA

Cluster Architecture



Host your application as container with automated fashion – k8s

ETCD : it's a DB that stores information in a key value pair

Install ETCD

1. Download Binaries

```
curl -L https://github.com/etcd-io/etcd/releases/download/v3.3.11/etcd-v3.3.11-linux-amd64.tar.gz -o etcd-v3.3.11-linux-amd64.tar.gz
```

2. Extract

```
tar xzvf etcd-v3.3.11-linux-amd64.tar.gz
```

3. Run ETCD Service

```
./etcd
```

I Operate ETCD

3. Run ETCD Service

```
./etcd
```

```
▶ ./etcdctl set key1 value1
```

```
▶ ./etcdctl get key1
```

```
value1
```

```
▶ ./etcdctl
```

```
NAME:  
  etcdctl - A simple command line client for etcd.  
COMMANDS:
```



Every information you see from the kubectl is FROM the ETCD server

Kubectl get node,pod,secret,configs,roles etc

Kube api server only component to interact with ETCD db

I Kube-api Server

1. Authenticate User

2. Validate Request

3. Retrieve data

4. Update ETCD

5. Scheduler

6. Kubelet

Kube scheduler only decides which pod has to deploy on which node , It doesnot actually place the pod in Node.Kubelet

deploys the pod in node

```
Compute-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-quota
  namespace: dev
spec:
  hard:
    pods: "10"
    requests.cpu: "4"
    requests.memory: 5Gi
    limits.cpu: "10"
    limits.memory: 10Gi
```

```
> kubectl create -f compute-quota.yaml
```

Schedular 5%

1. Manual schedule : Directly give node name

```
pod-definition.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 8080
  nodeName: node02
```

2. If pod is already deployed on the node but you want to put it in different node.

Then you have to give BIND POST request

Pod-bind-definition.yaml

```
apiVersion: v1
kind: Binding
metadata:
  name: nginx
target:
  apiVersion: v1
  kind: Node
  name: node02
```

```
▶ curl --header "Content-Type:application/json" --request POST --data '{"apiVersion":"v1", "kind": "Binding" ... }' http://$SERVER/api/v1/namespaces/default/pods/$PODNAME/binding/
```

Labels and selectors and annotations

K get pod –selector app=ov,wf=ov,hus=karthik

kubectl taint nodes master node-role.kubernetes.io/master:NoSchedule-

Resources and requirements and Limits

kubectl run nginx --image=nginx --restart=Never

--requests(cpu=100m,memory=256Mi) --limits(cpu=200m,memory=512Mi) --dry-run -o yaml

```
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
    - containerPort: 8080
  resources:
    requests:
      memory: "1Gi"
      cpu: 1
    limits:
      memory: "2Gi"
      cpu: 2
```

Container cannot use the more CPU than its limit. But memory can use than its limits.

Daemon Sets:

Daemon sets are like replicsets, It helps to run multiple instances of pods

But it run one copy of your pod in each node/cluster

It ensures one copy of pod is always runs in each node.

Best for monitoring and log views

Ex: kube-proxy pod in each node, wavenet in each node(Networking)

daemon-set-definition.yaml

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: monitoring-daemon
spec:
  selector:
    matchLabels:
      app: monitoring-agent
  template:
    metadata:
      labels:
        app: monitoring-agent
    spec:
      containers:
        - name: monitoring-agent
          image: monitoring-agent
```

replicaset-definition.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: monitoring-daemon
spec:
  selector:
    matchLabels:
      app: monitoring-agent
  template:
    metadata:
      labels:
        app: monitoring-agent
    spec:
      containers:
        - name: monitoring-agent
          image: monitoring-agent
```

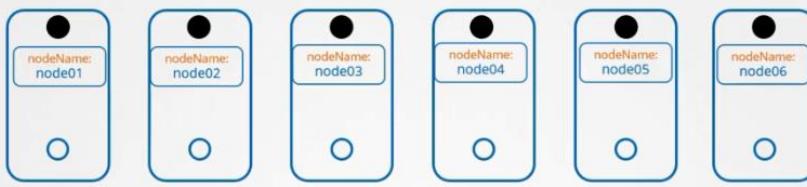
Set node name in the pod and deploy it in all node

I How does it work?



Default Behavior till v1.12

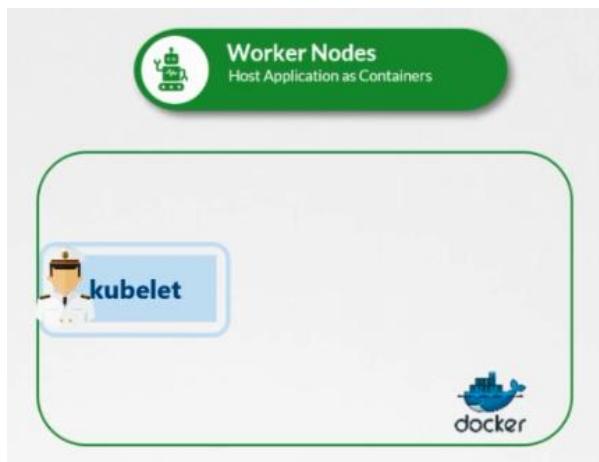
From v1.12 - uses NodeAffinity and default scheduler



Static pods:

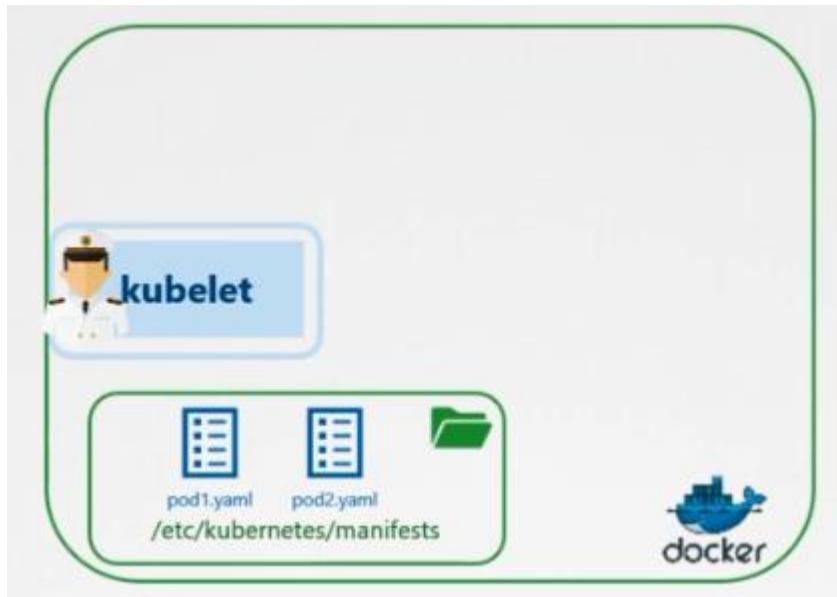
No master, no other nodes.

Only one node



Kubelet can manage to run th containers in the node without master.

Kubelet knows to create the pod. So it requires pod definition files. So place it in the specified directory



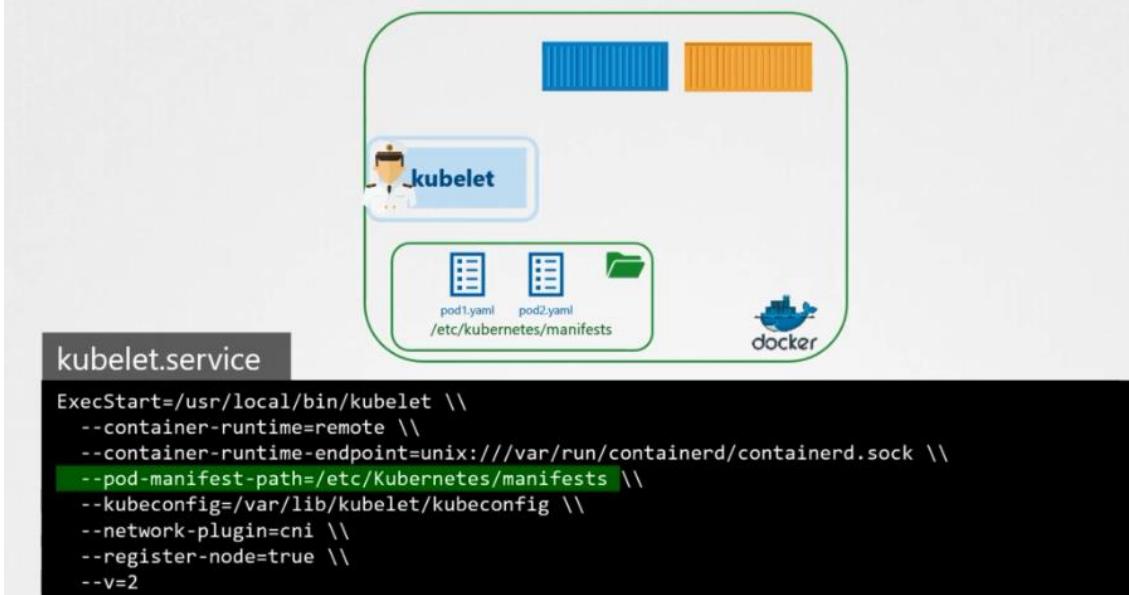
If any changes in the file, new file added/removed kubelet make sure the changes. It also ensure to rerun the pod if it fails.

We can only create the pod , not Deploy/replicaSet

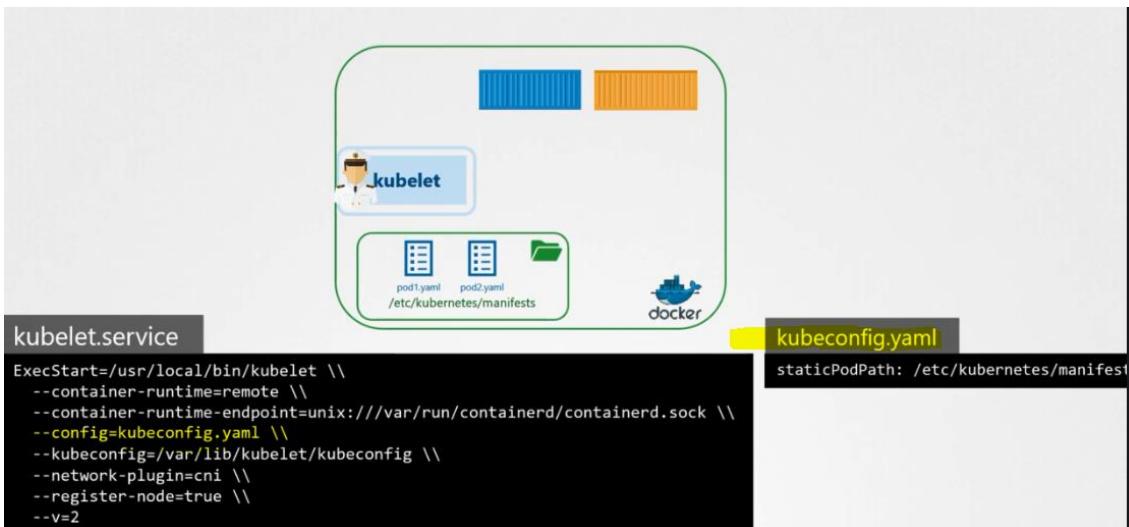
Kubelet can work only in the pod level

Directly you can specify the directory path like below or

I Static PODs



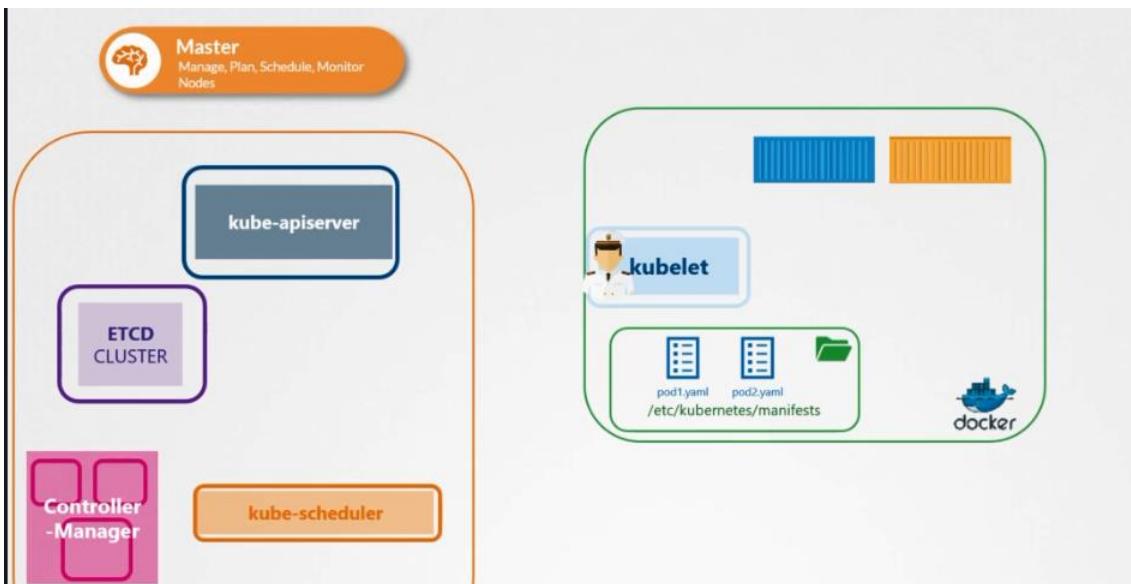
Specify common file and give the directory path in that file



```
ps -aux | grep kubelet
--config=/var/lib/kubelet/config.yaml
```

Kubectl works with kube-apiServer only. So with only kubelet we need to use docker ps.

It can also created pod from apiserver and local manifest file acting as static pod.

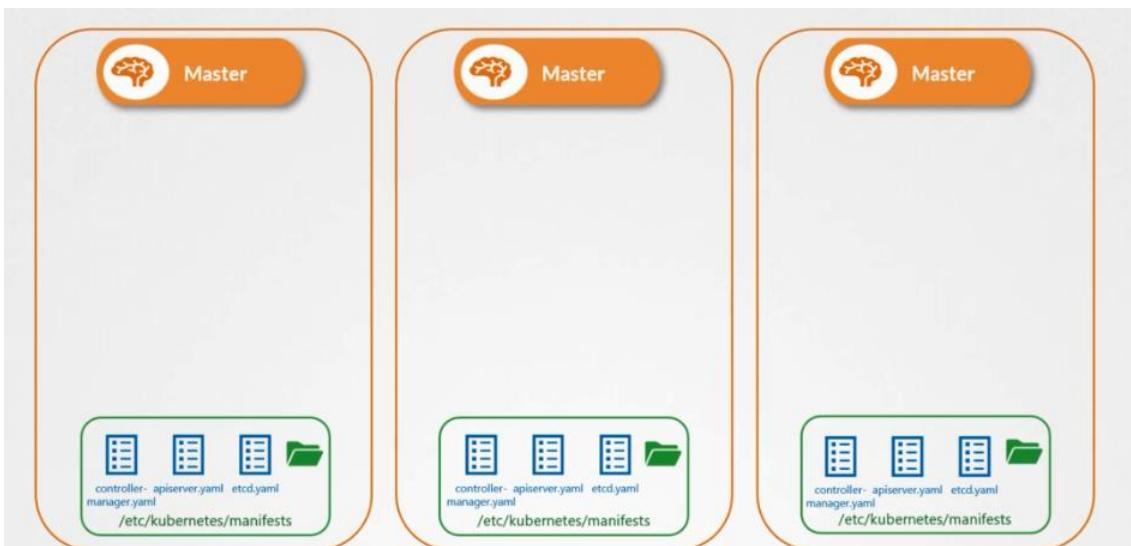


This time static pod also listed by kubectl

kubectl get pods					
NAME	READY	STATUS	RESTARTS	AGE	
static-web-node01	0/1	ContainerCreating	0	29s	

We can only read/view the pod. We cannot edit it from kube-api-server.

Static pod use case is making a node as master node.



Static PODs	DaemonSets
Created by the Kubelet	Created by Kube-API server (DaemonSet Controller)
Deploy Control Plane components as Static Pods	Deploy Monitoring Agents, Logging Agents on nodes
	Ignored by the Kube-Scheduler

Multiple Schedulers:

We can create your custom scheduler

| Deploy Additional Scheduler

```
▶ wget https://storage.googleapis.com/kubernetes-release/release/v1.12.0/bin/linux/amd64/kube-scheduler
kube-scheduler.service
ExecStart=/usr/local/bin/kube-scheduler \
--config=/etc/kubernetes/config/kube-scheduler.yaml \
--scheduler-name=default-scheduler

my-custom-scheduler.service
ExecStart=/usr/local/bin/kube-scheduler \
--config=/etc/kubernetes/config/kube-scheduler.yaml \
--scheduler-name=my-custom-scheduler
```

| Deploy Additional Scheduler - kubeadm

/etc/kubernetes/manifests/kube-scheduler.yaml	my-custom-scheduler.yaml
<pre>apiVersion: v1 kind: Pod metadata: name: kube-scheduler namespace: kube-system spec: containers: - command: - kube-scheduler - --address=127.0.0.1 - --kubeconfig=/etc/kubernetes/scheduler.conf - --leader-elect=true image: k8s.gcr.io/kube-scheduler-amd64:v1.11.3 name: kube-scheduler</pre>	<pre>apiVersion: v1 kind: Pod metadata: name: my-custom-scheduler namespace: kube-system spec: containers: - command: - kube-scheduler - --address=127.0.0.1 - --kubeconfig=/etc/kubernetes/scheduler.conf - --leader-elect=true - --scheduler-name=my-custom-scheduler - --lock-object-name=my-custom-scheduler image: k8s.gcr.io/kube-scheduler-amd64:v1.11.3 name: kube-scheduler</pre>

kubectl get pods --namespace=kube-system				
NAME	READY	STATUS	RESTARTS	AGE
coredns-78fcdf6894-bk4ml	1/1	Running	0	1h
coredns-78fcdf6894-ppr6m	1/1	Running	0	1h
etcd-master	1/1	Running	0	1h
kube-apiserver-master	1/1	Running	0	1h
kube-controller-manager-master	1/1	Running	0	1h
kube-proxy-dbgqv	1/1	Running	0	1h
kube-proxy-fptbr	1/1	Running	0	1h
kube-scheduler-master	1/1	Running	0	1h
my-custom-scheduler	1/1	Running	0	9s
weave-net-4tfpt	2/2	Running	1	1h
weave-net-6j6zs	2/2	Running	1	1h

```
pod-definition.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
  schedulerName: my-custom-scheduler
```

Kubectl get events command to list all the events in the current namespace

View Scheduler Logs

kubectl logs my-custom-scheduler --name-space=kube-system	
I0204 09:42:25.819338	1 server.go:126] Version: v1.11.3
W0204 09:42:25.822720	1 authorization.go:47] Authorization is disabled
W0204 09:42:25.822745	1 authentication.go:55] Authentication is disabled
I0204 09:42:25.822801	1 insecure_serving.go:47] Serving healthz insecurely on 127.0.0.1:10251
I0204 09:45:14.725487	1 controller_utils.go:1025] Waiting for caches to sync for scheduler controller
I0204 09:45:14.825634	1 controller_utils.go:1032] Caches are synced for scheduler controller
I0204 09:45:14.825814	1 leaderelection.go:185] attempting to acquire leader lease kube-system/my-custom-scheduler...
I0204 09:45:14.834953	1 leaderelection.go:194] successfully acquired lease kube-system/my-custom-scheduler

Logging and Monitoring

Monitors:

Node level :

No of nodes, healthy, performance, cpu , mem, disk utilization

pod level metrics :

no of pods, cpu, mem consumptions

no build in monitor in k8s

other property solutions

I Monitor

METRICS SERVER



Prometheus



Elastic Stack



DATADOG



dynatrace



minikube

```
▶ minikube addons enable metrics-server
```

others

```
▶ git clone https://github.com/kubernetes-incubator/metrics-server.git
```

```
▶ kubectl create -f deploy/1.8+/
clusterrolebinding "metrics-server:system:auth-delegator" created
rolebinding "metrics-server-auth-reader" created
apiservice "v1beta1.metrics.k8s.io" created
```

```
▶ kubectl top node
```

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
kubemaster	166m	8%	1337Mi	70%
kubenode1	36m	1%	1046Mi	55%
kubenode2	39m	1%	1048Mi	55%

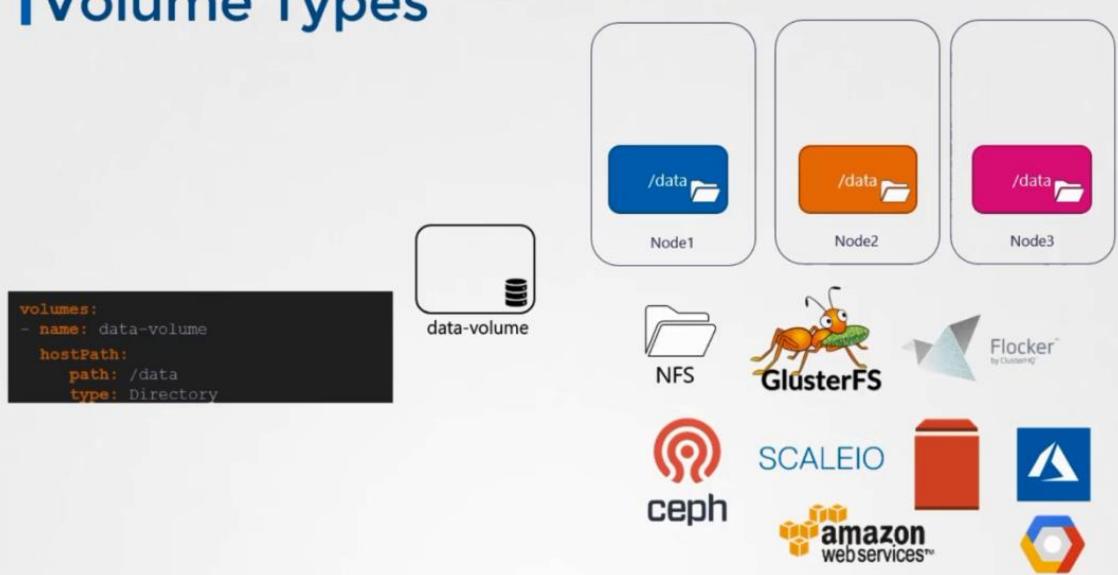
```
▶ kubectl top pod
```

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
nginx	166m	8%	1337Mi	70%
redis	36m	1%	1046Mi	55%

Kubectl logs -f pod-name container-name

Kubectl logs podname

I Volume Types



Application topic

Know deployments, rollingupdate,recreate,rollout

Must study config topic of CKAD

Command line args:

k run --image=kodekloud/webapp-color webapp-green --restart=Never -- --color green

ENV,config map, secrets, multi container, init container, self healing -> replicas

Cluster maintenance

```
▶ kubectl drain node-1
```

```
▶ kubectl cordon node-2
```

```
▶ kubectl uncordon node-1
```

kubeadm - upgrade



```
▶ kubeadm upgrade plan
```

```
[preflight] Running pre-flight checks.  
[upgrade] Making sure the cluster is healthy:  
[upgrade/config] Making sure the configuration is correct:  
[upgrade] Fetching available versions to upgrade to  
[upgrade/versions] Cluster version: v1.11.8  
[upgrade/versions] kubeadm version: v1.11.3  
[upgrade/versions] Latest stable version: v1.13.4  
[upgrade/versions] Latest version in the v1.11 series: v1.11.8
```

Components that must be upgraded manually after you have upgraded the control plane with 'kubeadm upgrade apply':

COMPONENT	CURRENT	AVAILABLE
Kubelet	3 x v1.11.3	v1.13.4

Upgrade to the latest stable version:

COMPONENT	CURRENT	AVAILABLE
API Server	v1.11.8	v1.13.4
Controller Manager	v1.11.8	v1.13.4
Scheduler	v1.11.8	v1.13.4
Kube Proxy	v1.11.8	v1.13.4
CoreDNS	1.1.3	1.1.3
Ftcd	3.2.18	N/A

kubeadm - upgrade



```
▶ apt-get upgrade -y kubeadm=1.12.0-00
```

```
▶ kubeadm upgrade apply v1.12.0
```

...

```
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.12.0". Enjoy!
```

```
[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading your kubelets if you haven't already done so.
```

```
▶ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	1d	v1.11.3
node-1	Ready	<none>	1d	v1.11.3
node-2	Ready	<none>	1d	v1.11.3



```
▶ apt-get upgrade -y kubelet=1.12.0-00
```

```
▶ systemctl restart kubelet
```

```
▶ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	1d	v1.12.0
node-1	Ready	<none>	1d	v1.11.3
node-2	Ready	<none>	1d	v1.11.3

Master node completed. Worker node upgrade



```
▶ kubectl drain node-1
```

```
▶ apt-get upgrade -y kubeadm=1.12.0-00
```

```
▶ apt-get upgrade -y kubelet=1.12.0-00
```

```
▶ kubeadm upgrade node config --kubelet-version v1.12.0
```

```
▶ systemctl restart kubelet
```

```
▶ kubectl uncordon node-1
```

After upgrading, We have to uncordon node to schedule a new pod in this node

I Backup - Resource Configs

kube-apiserver



Resource Configuration

```
▶ kubectl get all --all-namespaces -o yaml > all-deploy-services.yaml
```



VELERO

Formerly called ARK by HeptIO

I Backup - ETCD



ETCD Cluster



```
▶ ETCDCTL_API=3 etcdctl \
    snapshot save snapshot.db
```

```
▶ ls
snapshot.db
```

```
▶ ETCDCTL_API=3 etcdctl \
    snapshot status snapshot.db
```

HASH	REVISION	TOTAL KEYS	TOTAL SIZE
e63b3fc5	473353	875	4.1 MB

I Restore - ETCD



ETCD Cluster

```
▶ ETCDCCTL_API=3 etcdctl \
snapshot restore snapshot.db \
--data-dir /var/lib/etcd-from-backup \
--initial-cluster master-
1=https://192.168.5.11:2380,master-
2=https://192.168.5.12:2380 \
--initial-cluster-token etcd-cluster-1 \
--initial-advertise-peer-urls
https://${INTERNAL_IP}:2380

I | mvcc: restore compact to 475629
I | etcdserver/membership: added member 5e89ccdfc3
[https://192.168.5.12:2380] to cluster 894c7131f5165a78
I | etcdserver/membership: added member c8246cee7c
[https://192.168.5.11:2380] to cluster 894c7131f5165a78

▶ systemctl daemon-reload
▶ service etcd restart
```

```
▶ ETCDCCTL_API=3 etcdctl \
snapshot save snapshot.db

▶ ls
snapshot.db

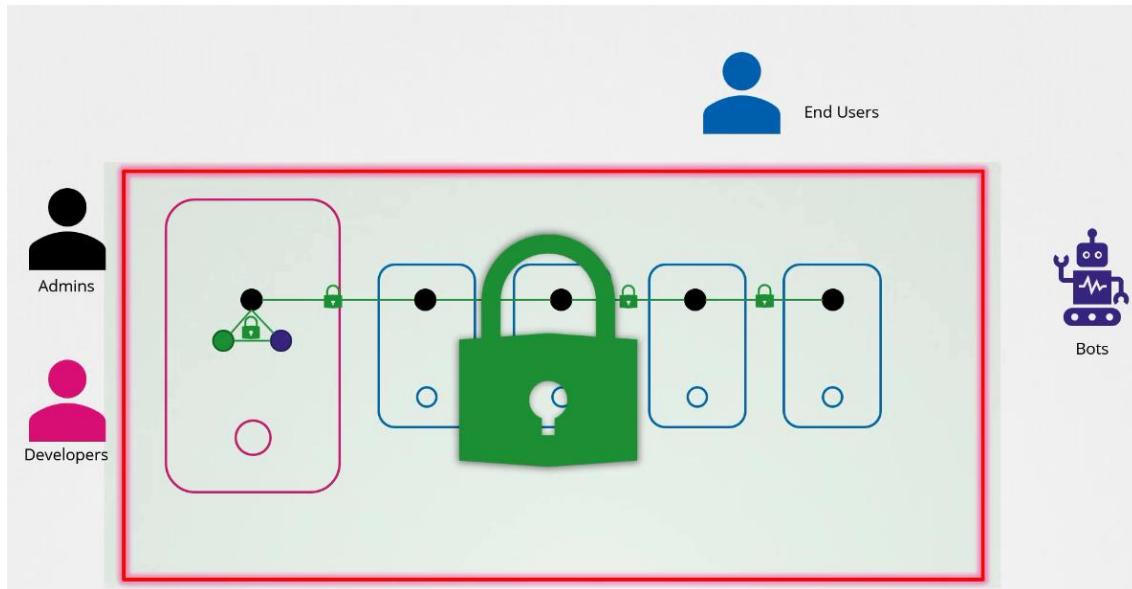
▶ service kube-apiserver stop
Service kube-apiserver stopped

[etcd.service]
ExecStart=/usr/local/bin/etcd \
--name ${ETCD_NAME} \
--cert-file=/etc/etcd/kubernetes.pem \
--key-file=/etc/etcd/kubernetes-key.pem \
--peer-cert-file=/etc/etcd/kubernetes.pem \
--peer-key-file=/etc/etcd/kubernetes-key.pem \
--trusted-ca-file=/etc/etcd/ca.pem \
--peer-trusted-ca-file=/etc/etcd/ca.pem \
--peer-client-cert-auth \
--client-cert-auth \
--initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \
--listen-peer-urls https://${INTERNAL_IP}:2380 \
--listen-client-urls https://${INTERNAL_IP}:2379,https://${INTERNAL_IP}:2380 \
--advertise-client-urls https://${INTERNAL_IP}:2379,https://${INTERNAL_IP}:2380 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster controller-0=https://${CONTROLLER_IP}:2380 \
--initial-cluster-state new \
--data-dir=/var/lib/etcd-from-backup
```

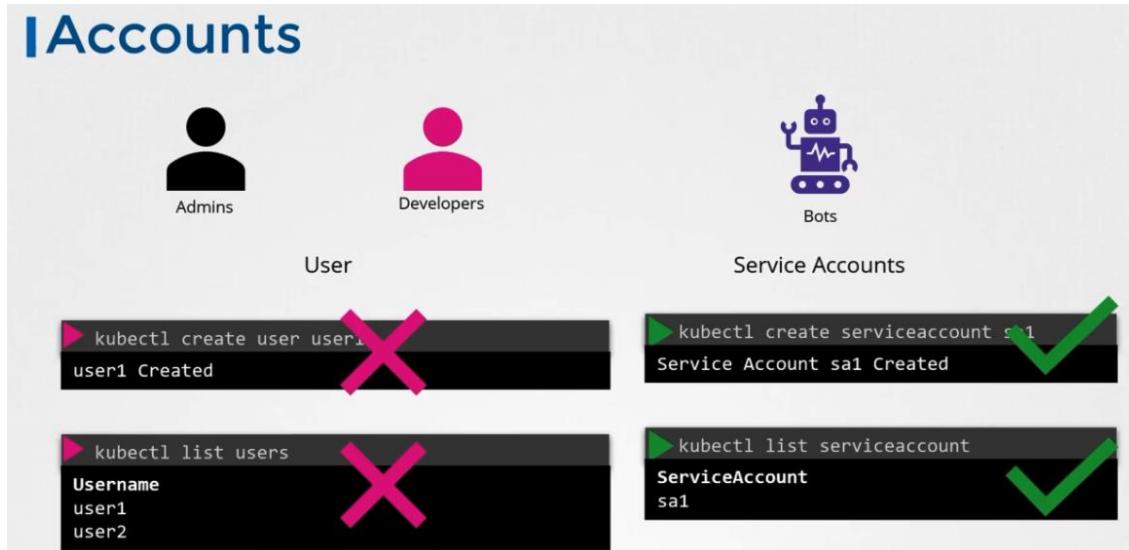
```
▶ service kube-apiserver start
Service kube-apiserver started
```

```
▶ ETCDCCTL_API=3 etcdctl \
snapshot save snapshot.db \
--endpoints=https://127.0.0.1:2379 \
--cacert=/etc/etcd/ca.crt \
--cert=/etc/etcd/etcd-server.crt \
--key=/etc/etcd/etcd-server.key
```

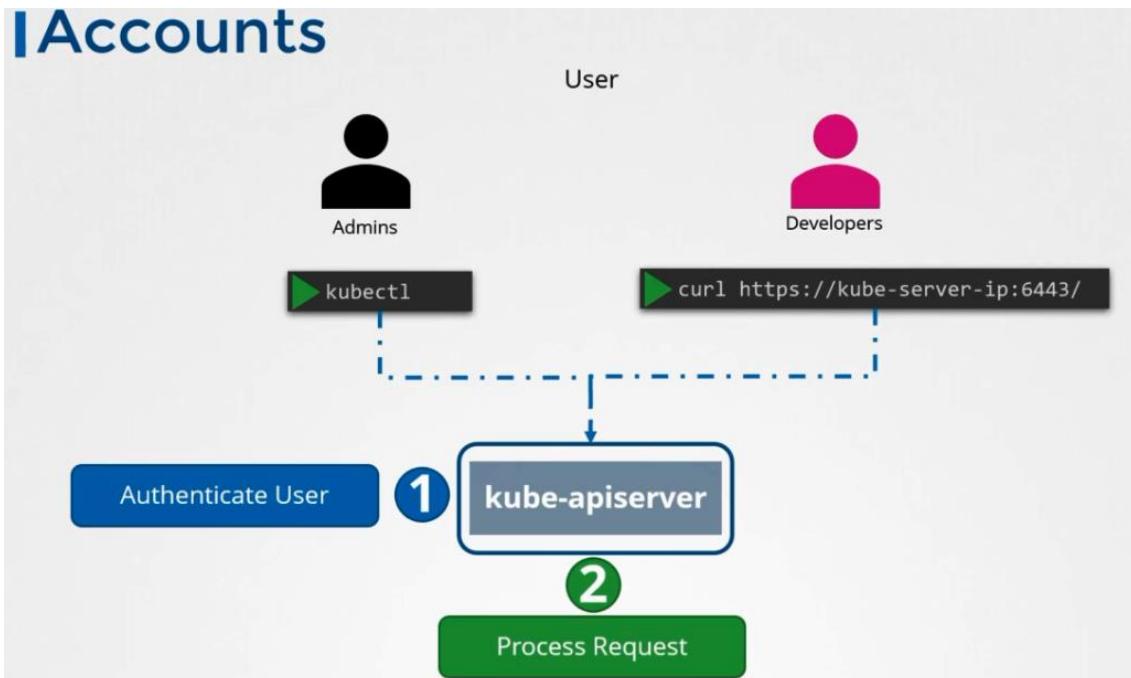
Security



K8s does not manage account natively . it depends on 3rd party like ldap.



I Accounts



How does kube-api server authenticate ?

I Auth Mechanisms



Static password File

```
user-details.csv
password123,user1,u0001
password123,user2,u0002
password123,user3,u0003
password123,user4,u0004
password123,user5,u0005
```

Pass the file name to the kubeapi server and restart it. For kubeadm -> automatically restarts the server

I Kube-api Server Configuration

manual *kubeadm*

```
kube-apiserver.service
ExecStart=/usr/local/bin/kube-apiserver \
--advertise-address=${INTERNAL_IP} \
--allow-privileged=true \
--apiserver-count=3 \
--authorization-mode=Node,RBAC \
--bind-address=0.0.0.0 \
--enable-swagger-ui=true \
--etcd-servers=https://127.0.0.1:2379 \
--event-ttl=1h \
--runtime-config=api/all \
--service-cluster-ip-range=10.32.0.0/24 \
--service-node-port-range=30000-32767 \
--v=2 \
--basic-auth-file=user-details.csv
```

```
/etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --authorization-mode=Node,RBAC
    - --advertise-address=172.17.0.107
    - --allow-privileged=true
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --basic-auth-file=user-details.csv
    image: k8s.gcr.io/kube-apiserver-amd64:v1.11.3
    name: kube-apiserver
```

Now you can access kube-api server with the provided authentication

I Authenticate User

```
curl -v -k https://master-node-ip:6443/api/v1/pods -u "user1:password123"
```

Similarly Token file also passed to kube api server, restart it. Authenticate the user

Static Token File

```
user-token-details.csv
KpjCVbI7rCFAHYPkByTlzRb7gu1cUc4B,user10,u0010,group1
rJjncHmvtxHc6M1WQddhtvNyyhgTdxSC,user11,u0011,group1
mjpoFIEiFOkL9toiKaRNtt59ePtczZSq,user12,u0012,group2
PG41TXhs7QjqwWkmBkvgGT9g1OyUq2ij,user13,u0013,group2
```

```
--token-auth-file=user-details.csv
```

```
curl -v -k https://master-node-ip:6443/api/v1/pods --header "Authorization: Bearer KpjCVbI7rCFAHYPkBzRb7gu1cUc4B"
```

Setup basic authentication on kubernetes

Note: This is not recommended in a production environment. This is only for learning purposes.

Follow the below instructions to configure basic authentication in a kubeadm setup.

Create a file with user details locally at `/tmp/users/user-details.csv`

```
1.      # User File Contents  
2.      password123,user1,u0001  
3.      password123,user2,u0002  
4.      password123,user3,u0003  
5.      password123,user4,u0004  
6.      password123,user5,u0005
```

Edit the kube-apiserver static pod configured by kubeadm to pass in the user details.

The file is located at `/etc/kubernetes/manifests/kube-apiserver.yaml`

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: kube-apiserver  
  namespace: kube-system  
spec:  
  containers:  
    - command:  
      - kube-apiserver  
        <content-hidden>  
    image: k8s.gcr.io/kube-apiserver-amd64:v1.11.3  
    name: kube-apiserver  
    volumeMounts:  
      - mountPath: /tmp/users  
        name: usr-details  
        readOnly: true  
    volumes:  
      - hostPath:  
          path: /tmp/users  
          type: DirectoryOrCreate  
        name: usr-details
```

Modify the kube-apiserver startup options to include the basic-auth file

```

apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
    - command:
      - kube-apiserver
      - --authorization-mode=Node,RBAC
      <content-hidden>
      - --basic-auth-file=/tmp/users/user-details.csv

```

Create the necessary roles and role bindings for these users:

```

---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "watch", "list"]

---
# This role binding allows "jane" to read pods in the "default" namespace
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
  - kind: User
    name: user1 # Name is case sensitive
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name of the Role or ClusterRole
  apiGroup: rbac.authorization.k8s.io

```

Once created, you may authenticate into the kube-api server using the users credentials

```
curl -v -k https://localhost:6443/api/v1/pods -u "user1:password123"
```

Encryption:

Symmetric encryption:

Same key used to encrypt and decrypt the data. So hackers easily get the key when you send to the server.

Asymteric:

It uses pair of keys . Private key and public key



Private Key

Public Key

For now we will think Public key as **public lock**

Privatekey only with the users



If you encrypt the data with public lock , you can only open it with associated private key

Create a private and public lock , send the public lock to the server.

ASYMMETRIC ENCRYPTION - SSH

```
▶ ssh-keygen  
id_rsa  id_rsa.pub
```



Private Key Public Lock

```
▶ cat ~/.ssh/authorized_keys  
ssh-rsa AAAAB3NzaC1yc..KhtUBfoTzlBqR  
V1NThvOo4opzEwRQo1mWx user1
```

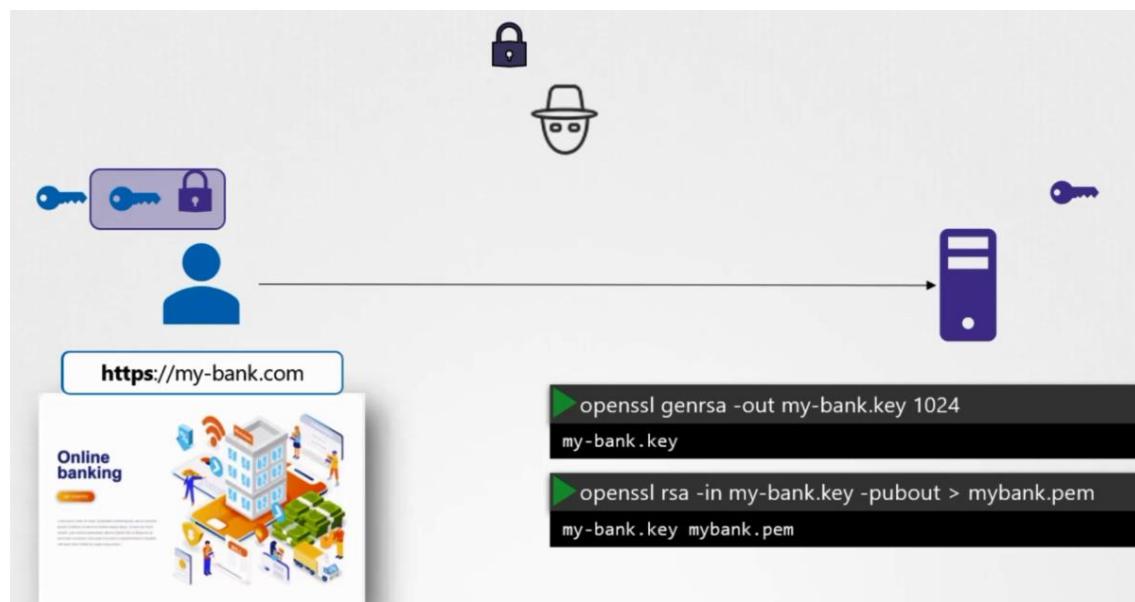


```
▶ ssh -i id_rsa user1@server1  
Successfully Logged In!
```

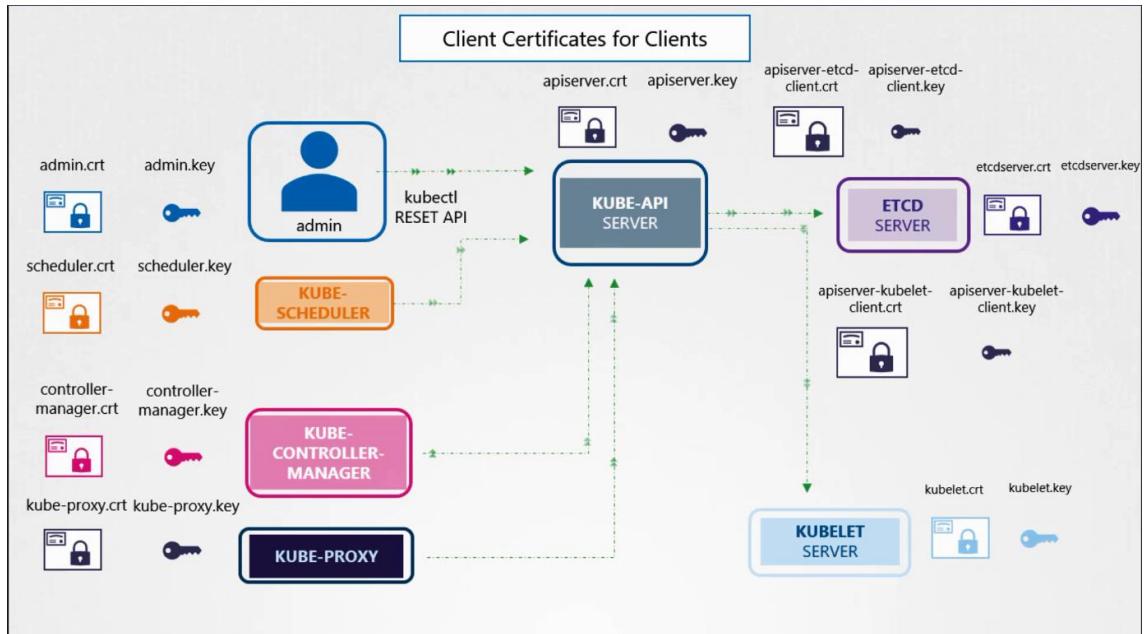
Generate private and public key at server. Send the public key to user.

Encrypt symmetric private key + public key & send to the server.

Now server can decrypt using its private key and get the users encrypted symmetric key. This works good . but we don't know that the server is valid are not. To ensure that its valid, we use CA certificate.



All the participants in the cluster must have unique identity (certificate and private key) to ensure and validate that they are valid users/systems



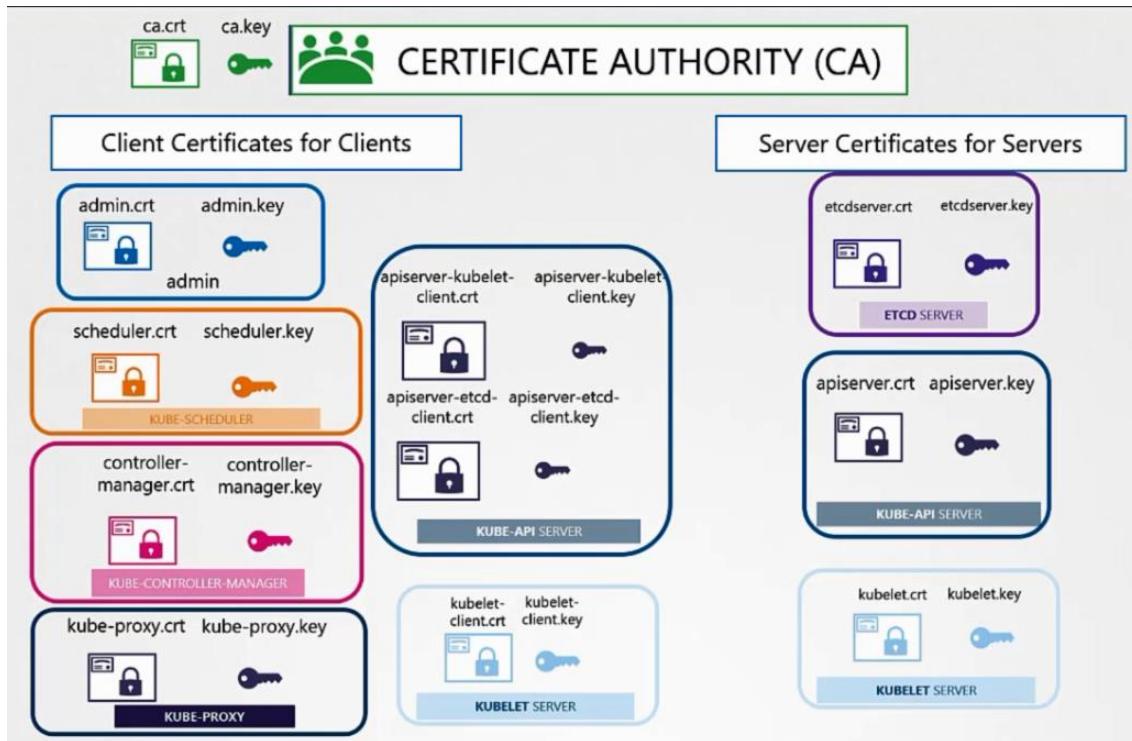
Creating certification : we can use the below tool to create certificate

EASYRSA

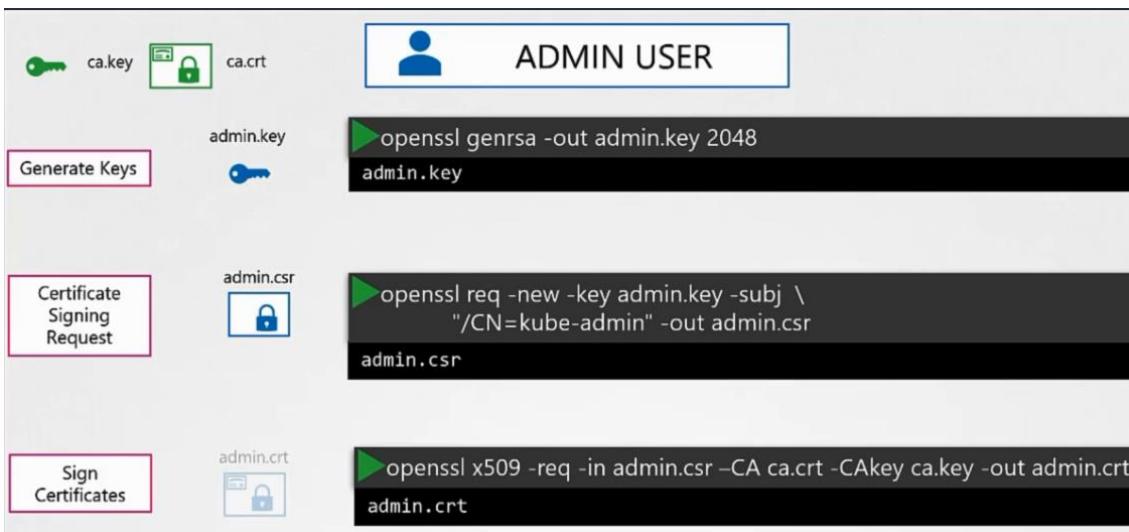
OPENSSL

CFSSL

Here we use openssl to generate certificate



Generating cert & key for CA



Admin users and other users are differentiated by the Groups . ex, admin is in system.masters group
It will be explained later

In the same way we create certificate for all the components.

What to do with the certificates?

For ex admin related tasks, send curl req with certificate.

```

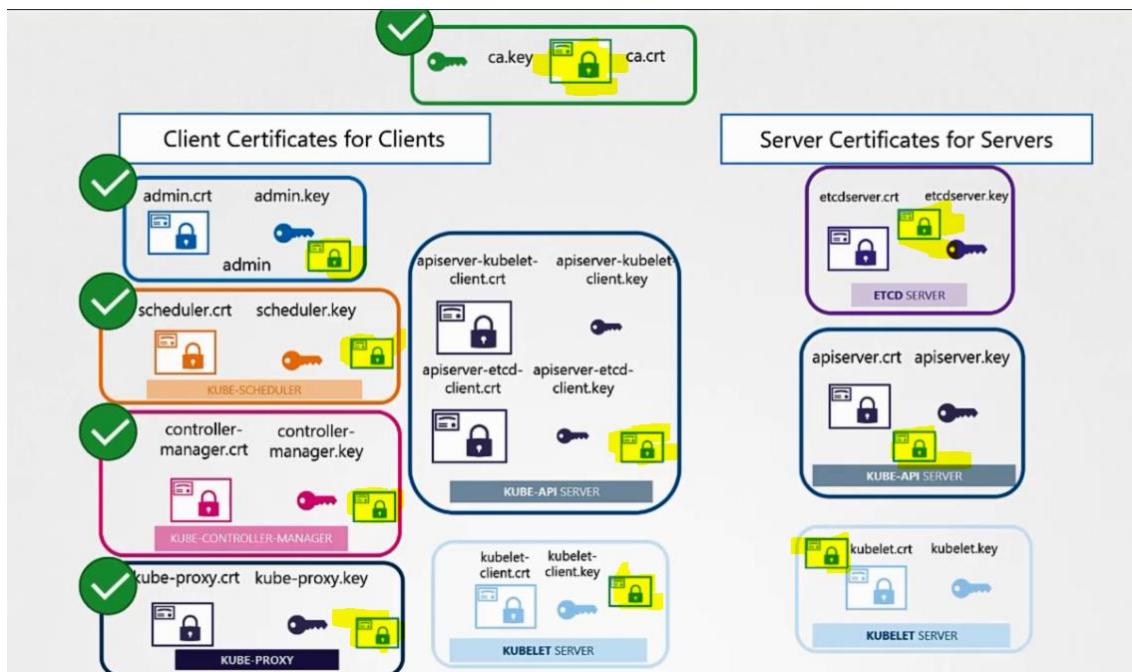
curl https://kube-apiserver:6443/api/v1/pods \
--key admin.key --cert admin.crt
--cacert ca.crt

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/pods",
  },
  "items": []
}

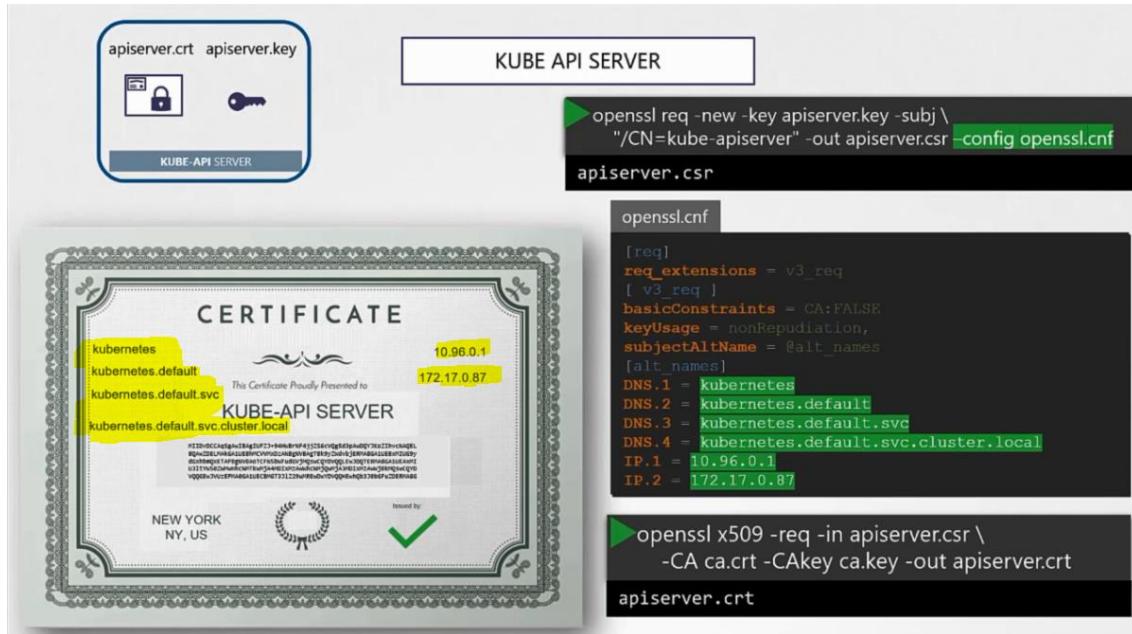
```

Every time we cannot type the certificate details in the command. So we mention it in kube-config file.

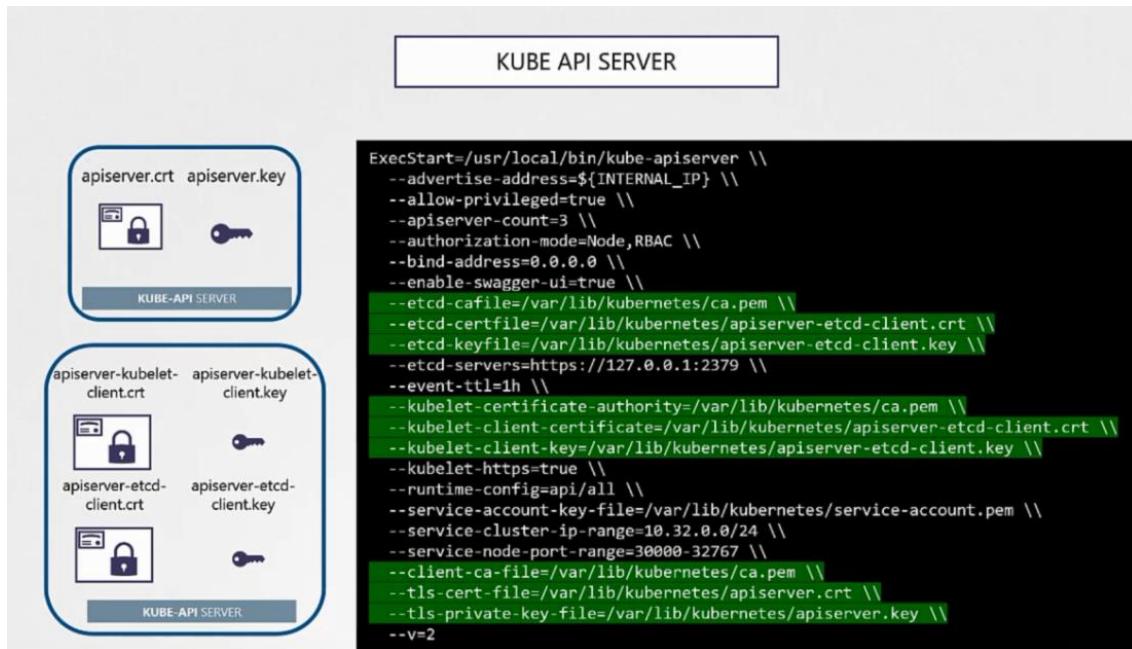
Whenever configure client or server in the cluster, we need to configure the CA certificate in all the components.



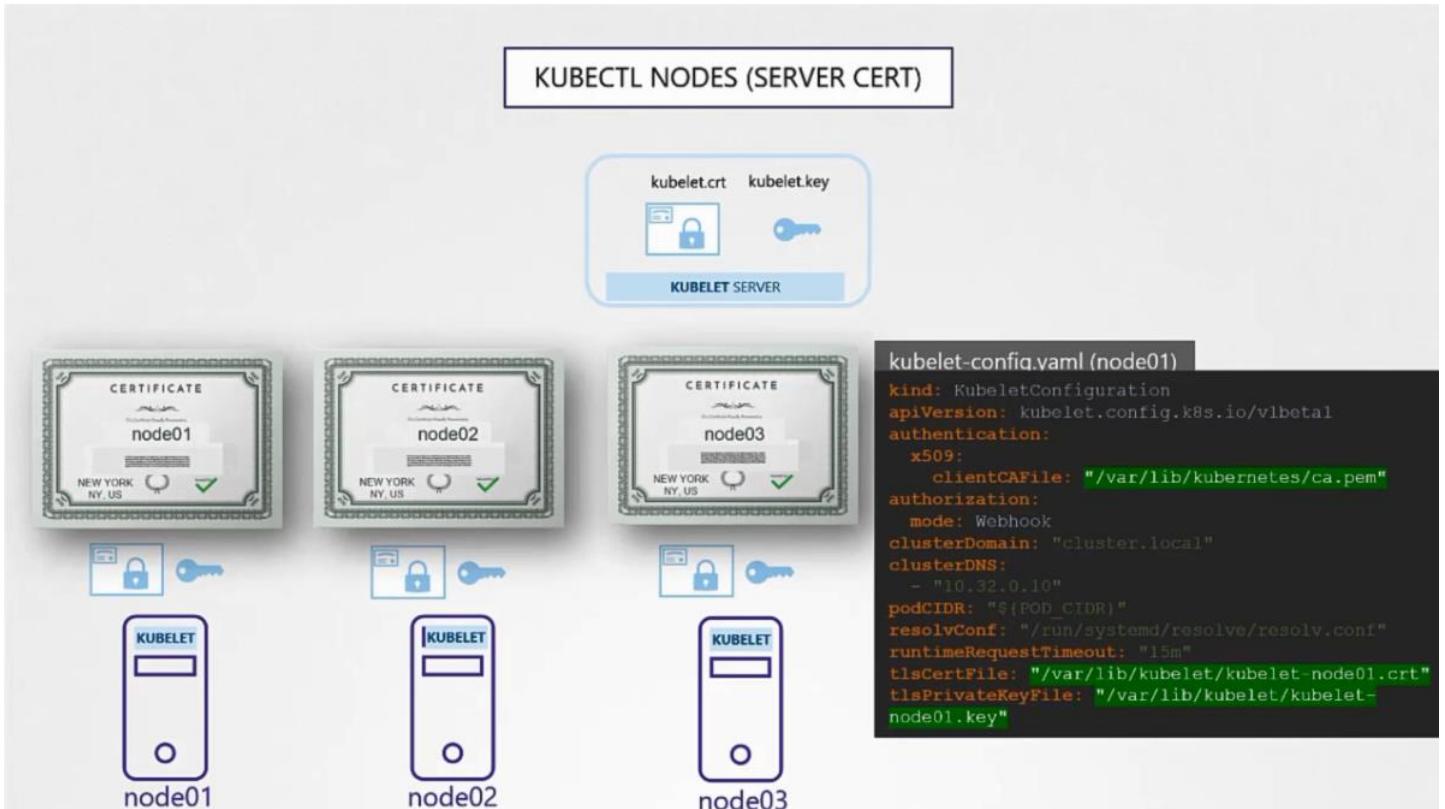
Kubeapi server is used most by all the components. It has many name, dns name, so multiple dns/ip can be configured as below.



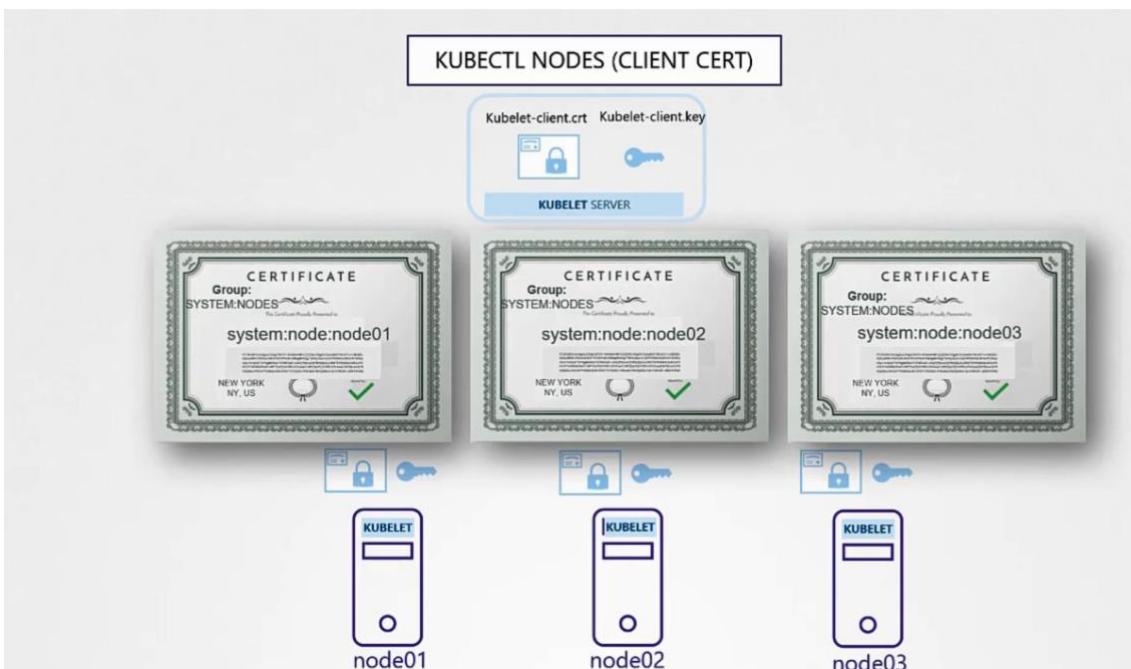
Kube-api server also act as client to etcd and kubelet . all the key should be mentioned as below



Kubelet running on each node. So certificates are issued to the nodename. These are server certificates.



Kubelet also have certificate on client side,



View certificates

1st know how k8s is installed

"The Hard Way"	kubeadm
<pre>▶ cat /etc/systemd/system/kube-apiserver.service</pre> <pre>[Service] ExecStart=/usr/local/bin/kube-apiserver \ --advertise-address=172.17.0.32 \ --allow-privileged=true \ --apiserver-count=3 \ --authorization-mode=Node,RBAC \ --bind-address=0.0.0.0 \ --client-ca-file=/var/lib/kubernetes/ca.pem \ --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 \ --event-ttl=1h \ --kubelet-certificate-authority=/var/lib/kubernetes/ca.pem \ --kubelet-client-key=/var/lib/kubernetes/kubernetes-key.pem \ --kubelet-https=true \ --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</pre>	<pre>▶ cat /etc/kubernetes/manifests/kube-apiserver.yaml</pre> <pre>spec: containers: - command: - kube-apiserver - --authorization-mode=Node,RBAC - --advertise-address=172.17.0.32 - --allow-privileged=true - --client-ca-file=/etc/kubernetes/pki/ca.crt - --disable-admission-plugins=PersistentVolumeLabel - --enable-admission-plugins=NodeRestriction - --enable-bootstrap-token-auth=true - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key - --etcd-servers=https://127.0.0.1:2379 - --insecure-port=0 - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key - --secure-port=6443 - --service-account-key-file=/etc/kubernetes/pki/sa.pub - --service-cluster-ip-range=10.96.0.0/12 - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key</pre>

In kubeadm :

```
▶ cat /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
spec:
  containers:
    - command:
        - kube-apiserver
        - --authorization-mode=Node,RBAC
        - --advertise-address=172.17.0.32
        - --allow-privileged=true
        - --client-ca-file=/etc/kubernetes/pki/ca.crt
        - --disable-admission-plugins=PersistentVolumeLabel
        - --enable-admission-plugins=NodeRestriction
        - --enable-bootstrap-token-auth=true
        - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
        - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
        - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
        - --etcd-servers=https://127.0.0.1:2379
        - --insecure-port=0
        - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
        - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
        - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
        - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
        - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
        - --secure-port=6443
        - --service-account-key-file=/etc/kubernetes/pki/sa.pub
        - --service-cluster-ip-range=10.96.0.0/12
        - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
        - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
```

```
/etc/kubernetes/pki/apiserver.crt
```

```
▶ openssl x509 -in /etc/kubernetes/pki/apiserver.crt -text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 3147495682089747350 (0x2bae26a58f090396)
    Signature Algorithm: sha256WithRSAEncryption
      Issuer: CN=kubernetes
      Validity
        Not Before: Feb 11 05:39:19 2019 GMT
        Not After : Feb 11 05:39:20 2020 GMT
      Subject: CN=kube-apiserver
      Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
          Public-Key: (2048 bit)
            Modulus:
              00:d9:69:38:80:68:3b:b7:2e:9e:25:00:e8:fd:01:
            Exponent: 65537 (0x10001)
      X509v3 extensions:
        X509v3 Key Usage: critical
          Digital Signature, Key Encipherment
        X509v3 Extended Key Usage:
          TLS Web Server Authentication
        X509v3 Subject Alternative Name:
          DNS:master, DNS:kubernetes, DNS:kubernetes.default,
          DNS:kubernetes.default.svc, DNS:kubernetes.default.svc.cluster.local, IP
          Address:10.96.0.1, IP Address:172.17.0.27
```

```
# Sign a new certificate for the apiserver-etcd-client
```

```
openssl x509 -req -in /etc/kubernetes/pki/apiserver-etcd-client.csr -CA /etc/kubernetes/pki/etcd/ca.crt -CAkey /etc/kubernetes/pki/etcd/ca.key -CAcreateserial -out /etc/kubernetes/pki/apiserver-etcd-client.crt
```

Certificate API:

CA server:

Master node is act as CA server.

When new admin asking for certificate, Admin has to login CA server, and issue certificate. After some time certificate will be expired and it has to renew,

Instead of login Ca server everytime, admin can use the API call to issue certificate



Flow

1. User first create the key

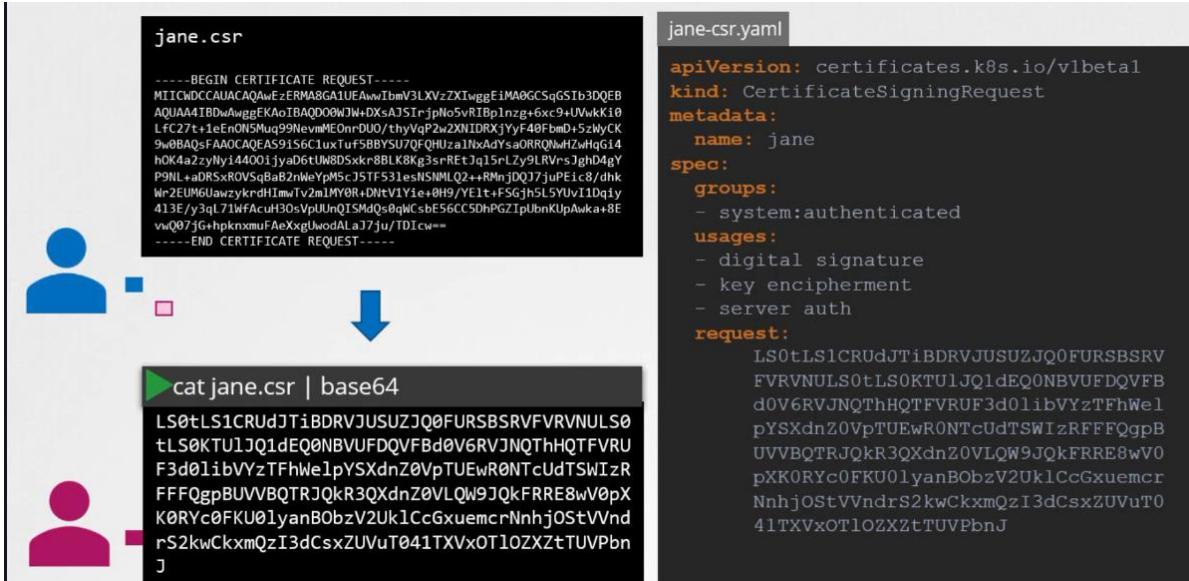
```
▶ openssl genrsa -out jane.key 2048
jane.key
```

2. Then generate the certificate signing request to admin

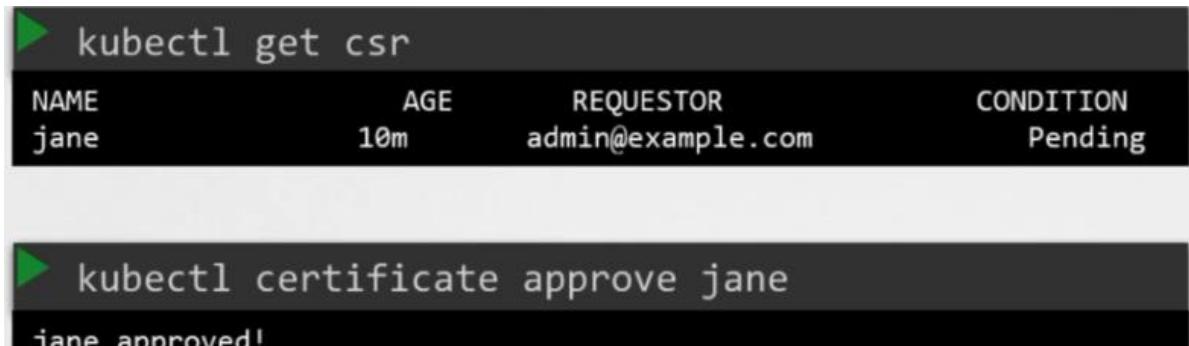
```
▶ openssl req -new -key jane.key -subj "/CN=jane" -out jane.csr
jane.csr

-----BEGIN CERTIFICATE REQUEST-----
MIICWDCCAUACAAQAwEzERMA8GA1UEAwIBmV3LXVzZXIwggEiMA0GCSqGSIb3DQE
AQAB4IBDwAwggEKAoIBAQD00WJW+DXsAJSIrjpNo5vRIBp1nwg+6xc9+UVwkKi0
LFC27t+1eEnON5Muq99NevmMEOnrDUO/thyVqP2w2XNIDRXjYyF40FbmD+5zWyCK
9w0BAQsFAAOCAQEAS91S6C1uxTuf5BBYSU7QFQHUza1NxAdYsaORRQNwHzwHqGi4
hOK4a2zyNy14400ijyaD6tUW8DSxkr8BLK8Kg3srRETJq15rLZy9LRVrsJghD4gY
P9NL+aDRSxROVSqBaB2nWeYpM5cJ5TF531esNSNMLQ2++RMnjDQJ7juPEic8/dh
Wr2EUM6UawzykrdHImwTv2m1MY0R+DNTV1Yie+0H9/YElt+FSGjh5LYUv1lDqiy
413E/y3ql71WFAcuH3OsVpUUnQISMdQs0qWCsbE56CC5DhPGZIpUbnKUpAwka+8E
vvQ07jG+hpknxmuFAeXxgUwodALaJ7ju/TDIcw==
-----END CERTIFICATE REQUEST-----
```

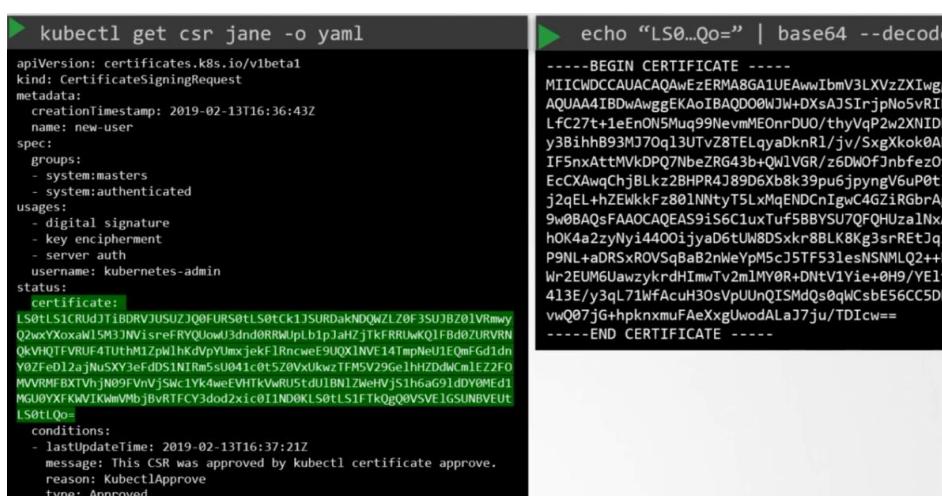
3. Admin takes the key, and create the certificate signing request object



4. Once certification sigining request is submitted, admin can use the command to check it and approve it



5. K8s sign the certificate using the CA key pair, and generate the certificate for the user. This certificate and then extracted and shared to the user



6. Control manager is responsible for CA related tasks



If anyone is signing the certificate, they should need ROOT certificate. Controller manager has it

```
▶ cat /etc/kubernetes/manifests/kube-controller-manager.yaml
spec:
  containers:
    - command:
        - kube-controller-manager
        - --address=127.0.0.1
        - --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt
        - --cluster-signing-key-file=/etc/kubernetes/pki/ca.key
        - --controllers=*,bootstrapsigner,tokencleaner
        - --kubeconfig=/etc/kubernetes/controller-manager.conf
        - --leader-elect=true
        - --root-ca-file=/etc/kubernetes/pki/ca.crt
        - --service-account-private-key-file=/etc/kubernetes/pki/sa.key
        - --use-service-account-credentials=true
```

KubeConfigs

```
▶ curl https://my-kube-playground:6443/api/v1/pods \
  --key admin.key
  --cert admin.crt
  --cacert ca.crt
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/pods",
  },
  "items": []
}
```



```
▶ kubectl get pods
--server my-kube-playground:6443
--client-key admin.key
--client-certificate admin.crt
--certificate-authority ca.crt
No resources found.
```

Typing those certificates every time is tedious tasks. So we kept it in kube config file.

```
$HOME/.kube/config
```

KubeConfig File

```
--server my-kube-playground:6443  
--client-key admin.key  
--client-certificate admin.crt  
--certificate-authority ca.crt
```

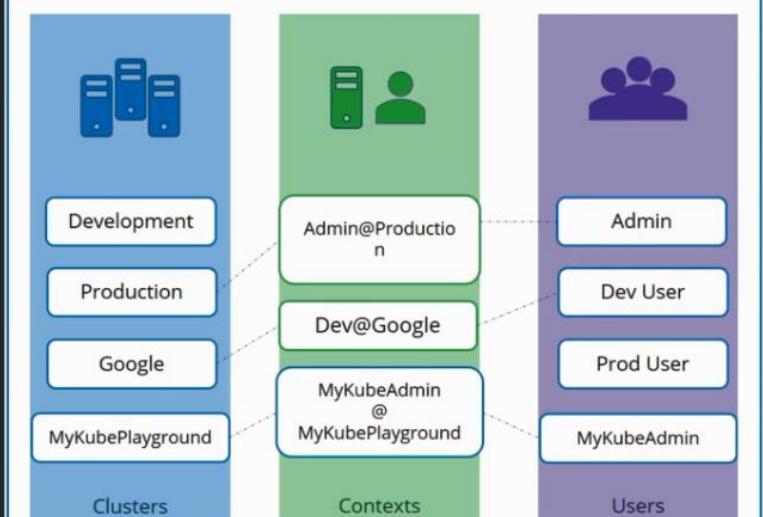
Config file has 3 sections. 1. Cluster 2. Contexts 3.Users

Context is bridge the cluster and Users. Providing access to which user has to use which cluster

KubeConfig File

```
apiVersion: v1  
kind: Config  
  
clusters:  
- name: my-kube-playground  
  cluster:  
    certificate-authority: ca.crt  
    server: https://my-kube-playground:6443  
  
contexts:  
- name: my-kube-admin@my-kube-playground  
  context:  
    cluster: my-kube-playground  
    user: my-kube-admin  
  
users:  
- name: my-kube-admin  
  user:  
    client-certificate: admin.crt  
    client-key: admin.key
```

```
$HOME/.kube/config
```



You can also mention the current context

```
apiVersion: v1  
kind: Config  
  
current-context: dev-user@google
```

I Kubectl config

```
▶ kubectl config view
apiVersion: v1
kind: Config
current-context: my-kube-admin@my-kube-playground

clusters:
- name: my-kube-playground
- name: development
- name: production

contexts:
- name: my-kube-admin@my-kube-playground
- Name: prod-user@production

users:
- name: my-kube-admin
- name: prod-user
```

```
▶ kubectl config use-context prod-user@production
apiVersion: v1
kind: Config
current-context: prod-user@production

clusters:
- name: my-kube-playground
- name: development
- name: production

contexts:
- name: my-kube-admin@my-kube-playground
- Name: prod-user@production

users:
- name: my-kube-admin
- name: prod-user
```

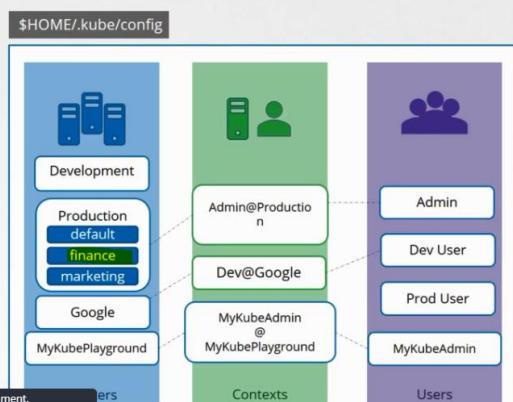
Namespaces

```
apiVersion: v1
kind: Config

clusters:
- name: production
  cluster:
    certificate-authority: ca.crt
    server: https://172.17.0.51:6443

contexts:
- name: admin@production
  context:
    cluster: production
    user: admin
    namespace: finance

users:
- name: admin
  user:
```



Certificates in KubeConfig

```
apiVersion: v1
kind: Config

clusters:
- name: production
  cluster:
    certificate-authority: /etc/kubernetes/pki/ca.crt
    server: https://172.17.0.51:6443

contexts:
- name: admin@production
  context:
    cluster: production
    user: admin
    namespace: finance

users:
- name: admin
  user:
    client-certificate: /etc/kubernetes/pki/users/admin.crt
    client-key: /etc/kubernetes/pki/users/admin.key
```

Use full path of the certification or u can paste the certificae

directly

Certificates in KubeConfig

```
apiVersion: v1
kind: Config

clusters:
- name: production
  cluster:
    certificate-authority: /etc/kubernetes/pki/ca.crt

    certificate-authority-data: LS0tLS1CRUdJTiBDRVJU
      SUZJQ0FURSBRSRVFVRVNULS0tLS0KTUlJ
      Q1dEQ0NBVUFDQVFBD0V6RVJNQThHQTfV
      RUF3d0libVYzTFhWe1pYSXdnZ0VpTUEw
      R0NTcUdTSWIzRFFFQgpBUVVBQTRJQkR3
      QXdnZ0VLQW9JQkFRRE8wV0pXK0RYc0FK
      U01yanB0bzV2Uk1CcGxuemcrNnhj0StV
      VndrS2kwCkxmQzI3dCsxZUVuT041TXVx
      OTl0ZXztTUVPbnJ
```

```
-----BEGIN CERTIFICATE-----
MIICWDCCAUACQAwEzERMA8GA1UEAwwIbmV3LXVzZXIxggEiMA0G
AQAA4IBDwAwggEKAoIBAQD00WJW+DXsAJS1rjpNo5vR1Bp1nZg+
Lfc27t+1eEnONSUug99NevmMEOnrDUO/thyVqP2w2XNIDRxjYyF46
y3BihhB93MJ70q13UTVz8TELqyaDknR1/jv/SxgXkok0ABUTpwMx4
IFNxattMVKdPQ7NbeZRG43b+QW1VGR/z6DWOfnbfezoTaAydgL1
EcCXAwqChjBLkz2BHPR4J89D6xb8k39pu6jpyngV6uP0tIb0zpqn
j2qEL+hZEWkkFz801NtyT5LxhqENDCnIgwC4GZirGbrAgIBAAgg/
9wBAQsFAAOCAQEAS9i56C1uxTuf5BBYSU7QFQHUzalNxAdysaORF
hOK4a2zyNy14400ijyaD6tUW8DSxkr8BLK8kg3srRetJql5rLzy9l
P9NL+aDRSxROV5gBaB2nWeYpM5cJ5TF531esNSNMLQ2+rRMnjDQJ;
Wr2E6UawzyKrdHImwTv2m1MY0R+DNTv1Yie+0H9/YEl1t+FSGjh:
413E/y3qL71wfAcuh3OsVpUUuQISMdQs0qWcsbE56CC5DhPGZiPut
vw07jG+hpknxmuFAeXxgUwodAlaJ7ju/TDIcw==

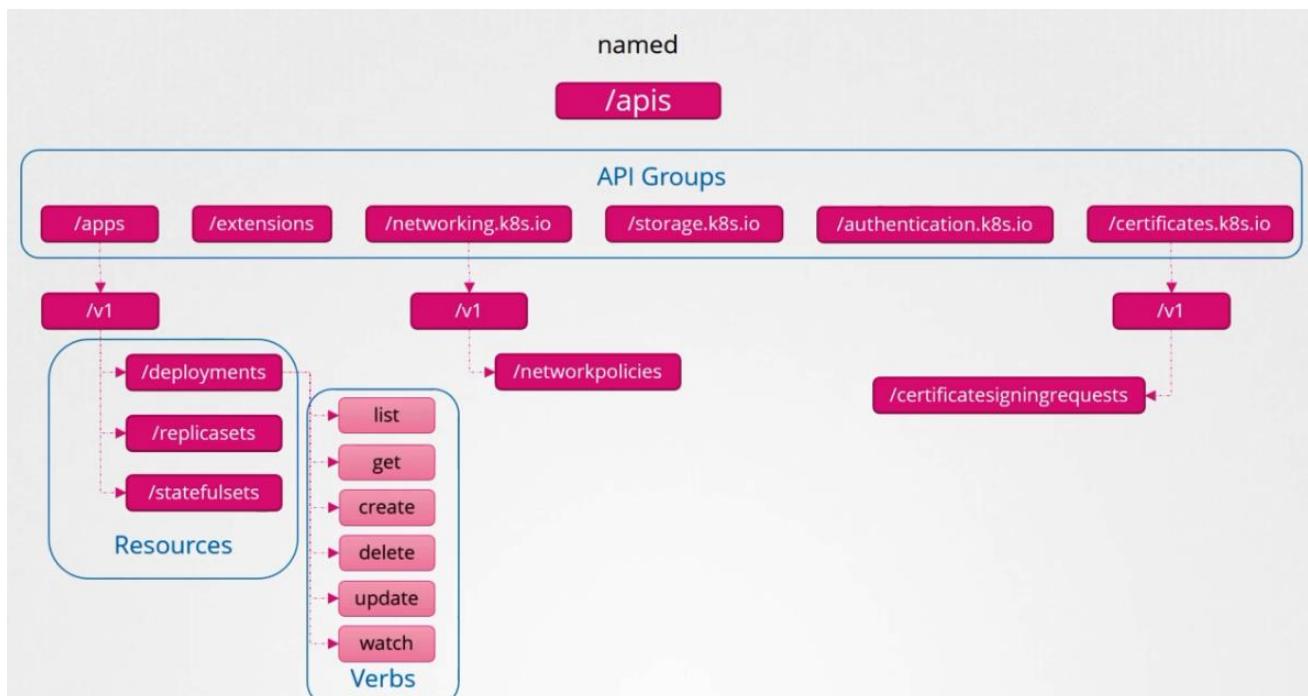
-----END CERTIFICATE-----
```

cat ca.crt | base64

```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBRSRVFVRN
tLS0KTUlJQ1dEQ0NBVUFDQVFBD0V6RVJNQThHQT
F3d0libVYzTFhWe1pYSXdnZ0VpTUEwR0NTcUdTS
FFFQgpBUVVBQTRJQkR3QXdnZ0VLQW9JQkFRRE8w
K0RYc0FKU01yanB0bzV2Uk1CcGxuemcrNnhj0StV
rS2kwCkxmQzI3dCsxZUVuT041TXVxOTl0ZXztTU
J
```

```
kubectl config --kubeconfig=/root/my-kube-config use-context research
```

Api Groups:



No Access



Kube ApiServer

```
curl http://localhost:6443 -k
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"\"",
  "reason": "Forbidden",
  "details": {
    "code": 403
}
```

Because certificates required

```
curl http://localhost:6443 -k
--key admin.key
--cert admin.crt
--cacert ca.crt
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/healthz",
    "/logs",
```

kubectl proxy



Kubectl Proxy



Kube ApiServer



```
kubectl proxy
```

```
Starting to serve on 127.0.0.1:8001
```

```
curl http://localhost:8001 -k
```

```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/healthz",
    "/logs",
    "/metrics",
    "/openapi/v2",
    "/swagger-2.0.0.json",
```

Kube proxy



Kubectl proxy

Kubectl – http proxy to access the kube api server; kube proxy – responsible to connect all the pods/services

Role Based Access Control

IRBAC

The diagram illustrates a developer role. On the left, a pink crown icon is above a rounded rectangle containing a bulleted list of permissions: Can view PODs, Can create PODs, Can Delete PODs, and Can Create ConfigMaps. Below this list is the word "Developer". On the right, a code editor window shows the YAML configuration for the "developer-role.yaml" file. The file defines a Role named "developer" with rules for the "pods" and "ConfigMap" resources across all API groups, allowing list, get, create, update, delete, and create actions respectively.

```
developer-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer
rules:
- apiGroups: []
  resources: ["pods"]
  verbs: ["list", "get", "create", "update", "delete"]
- apiGroups: []
  resources: ["ConfigMap"]
  verbs: ["create"]
```

kubectl create -f developer-role.yaml

The diagram illustrates a developer binding. On the left, a pink user icon is followed by a green arrow pointing to a pink crown icon above a rounded rectangle containing the same permission list as the previous diagram. Below this list is the word "Developer". Below the developer section is the text "Namespace: default". On the right, two code editor windows are shown. The top window shows the "developer-role.yaml" file, identical to the one above. The bottom window shows the "devuser-developer-binding.yaml" file, which defines a RoleBinding named "devuser-developer-binding" that maps the "dev-user" user to the "developer" role in the "rbac.authorization.k8s.io" API group.

```
developer-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer
rules:
- apiGroups: []
  resources: ["pods"]
  verbs: ["list", "get", "create", "update", "delete"]
- apiGroups: []
  resources: ["ConfigMap"]
  verbs: ["create"]
```

```
devuser-developer-binding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: devuser-developer-binding
subjects:
- kind: User
  name: dev-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: developer
  apiGroup: rbac.authorization.k8s.io
```

kubectl create -f devuser-developer-binding.yaml

IView RBAC

```
▶ kubectl get roles
```

NAME	AGE
developer	4s

```
▶ kubectl get rolebindings
```

NAME	AGE
devuser-developer-binding	24s

```
▶ kubectl describe role developer
```

Name:	developer		
Labels:	<none>		
Annotations:	<none>		
PolicyRule:			
Resources	Non-Resource URLs	Resource Names	Verbs
ConfigMap	[]	[]	[create]
pods	[]	[]	[get watch list create delete]

```
▶ kubectl describe rolebinding devuser-developer-binding
```

Name:	devuser-developer-binding	
Labels:	<none>	
Annotations:	<none>	
Role:		
Kind:	Role	
Name:	developer	
Subjects:		
Kind	Name	Namespace
User	dev-user	

Check Access

```
▶ kubectl auth can-i create deployments  
yes
```

```
▶ kubectl auth can-i delete nodes  
no
```

```
▶ kubectl auth can-i create deployments --as dev-user  
no
```

```
▶ kubectl auth can-i create pods --as dev-user  
yes
```

```
▶ kubectl auth can-i create pods --as dev-user --namespace test  
no
```

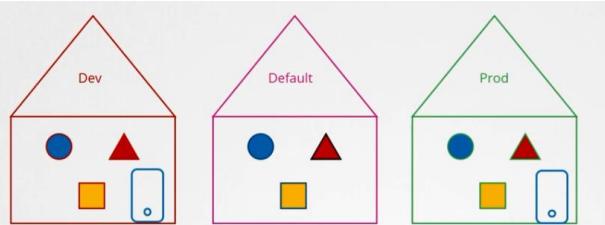
Pod level restrictions to the developer user

| Resource Names

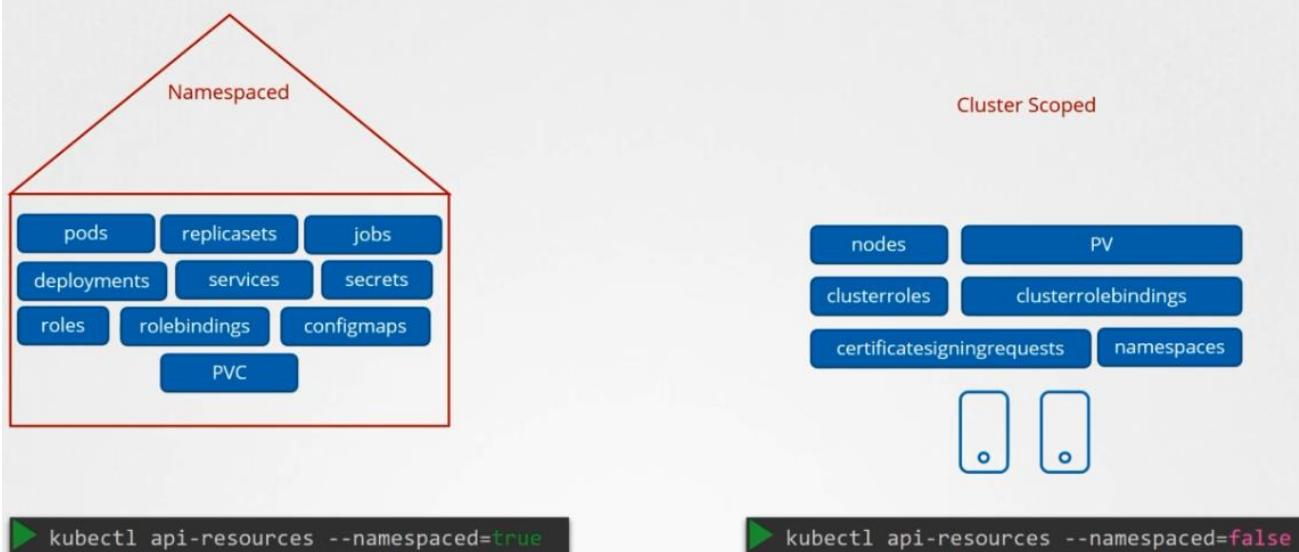


Cluster roles and role bindings:

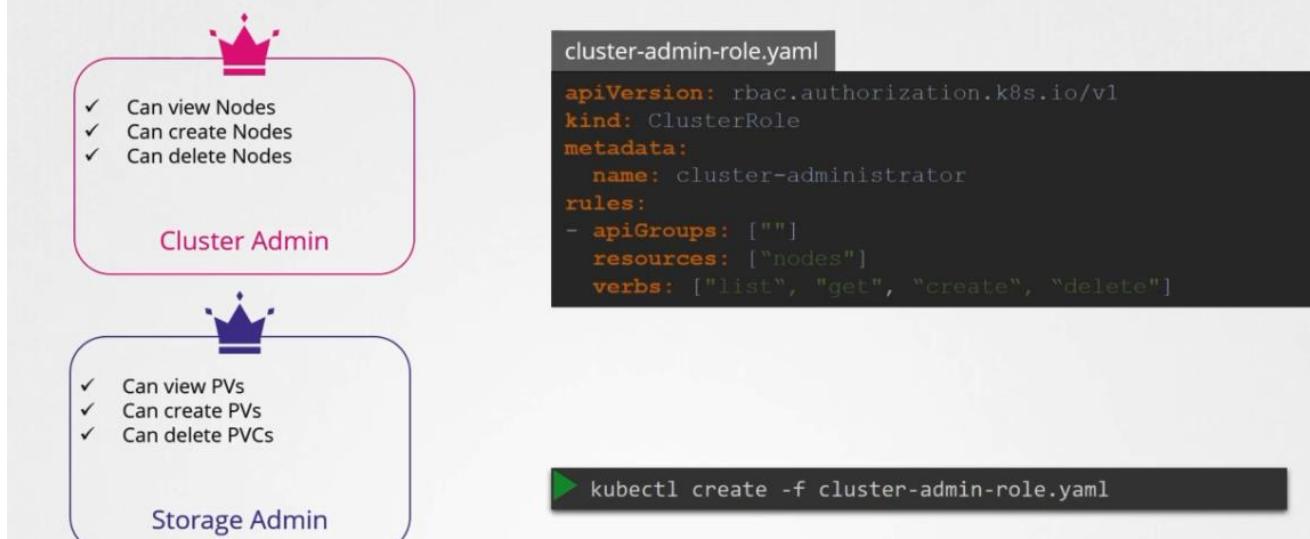
Nodes are cluster scope , it cannot be bind with namespaces



| Namespace



clusterroles



clusterrolebinding



```
cluster-admin-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-administrator
rules:
- apiGroups: []
  resources: [nodes]
  verbs: [list, get, create, delete]
```

```
cluster-admin-role-binding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-role-binding
subjects:
- kind: User
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-administrator
  apiGroup: rbac.authorization.k8s.io
```

Cluster roles can be applied to all namespace resources.

Secure images : private repo

image: docker.io/nginx/nginx

The diagram shows the components of a Docker image URL. The string "docker.io/nginx/nginx" is split into three parts by curly braces: "docker.io" (Registry), "nginx" (User/Account), and "nginx" (Image/Repository).

IPrivate Repository

```
▶ docker login private-registry.io
```

```
▶ docker run private-registry.io/apps/internal-app
```

```
nginx-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: private-registry.io/apps/internal-app
  imagePullSecrets:
  - name: regcred
```

```
▶ kubectl create secret docker-registry regcred \
--docker-server= private-registry.io \
--docker-username= registry-user \
--docker-password= registry-password \
--docker-email= registry-user@org.com
```

Security contexts, Network policy – from ckad

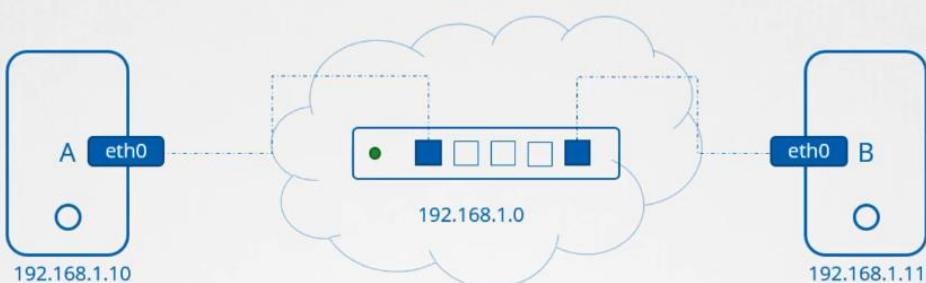
Networking

I Networking Pre-Requisites

- Switching and Routing
 - Switching
 - Routing
 - Default Gateway
- DNS
 - DNS Configurations on Linux
 - CoreDNS Introduction
- Network Namespaces
- Docker Networking

Switching helps to connect internal computers

I Switching



1 ➤ ip link
eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000

2 ➤ ip addr add 192.168.1.10/24 dev eth0

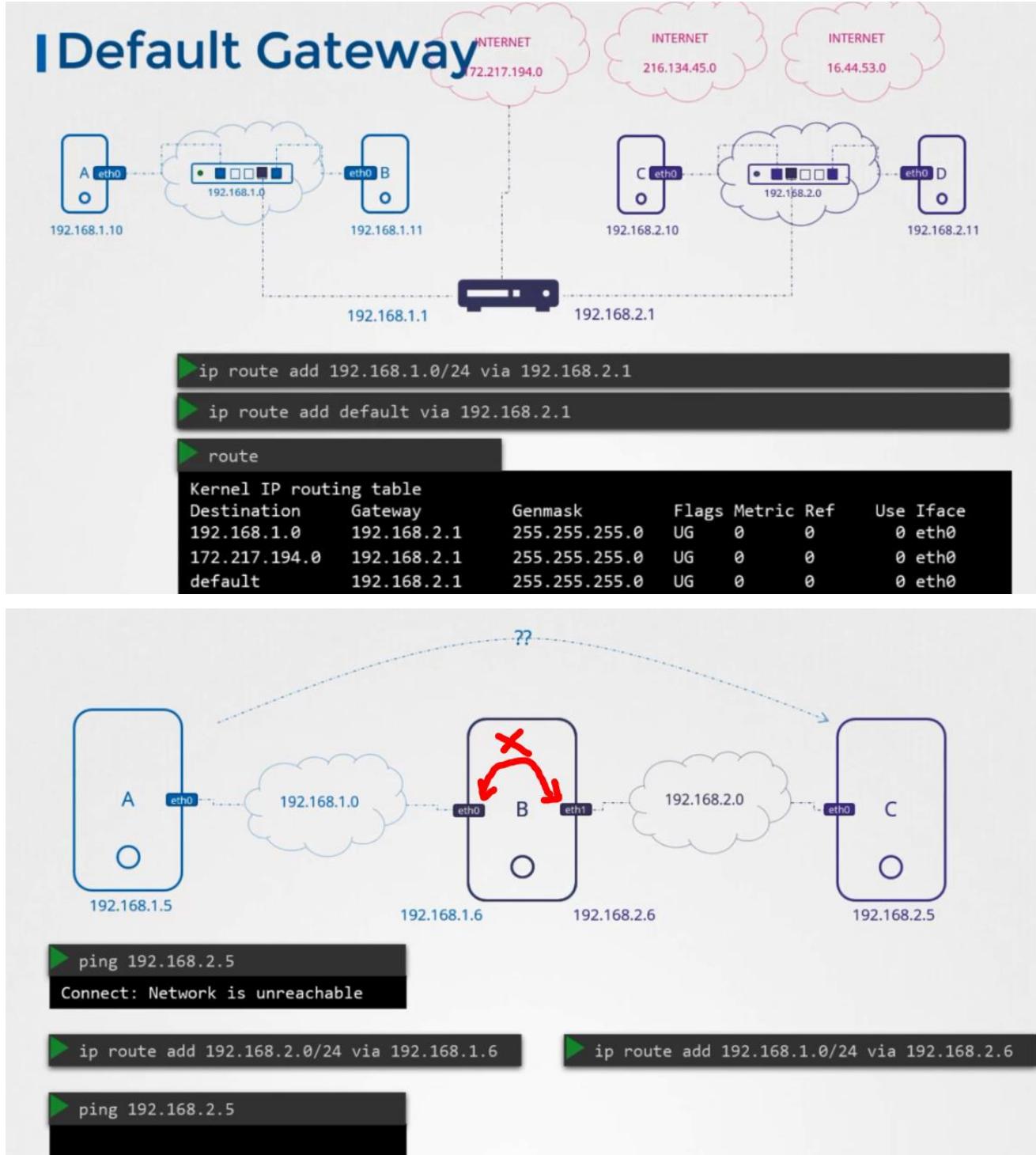
➤ ip link
eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000

➤ ip addr add 192.168.1.11/24 dev eth0

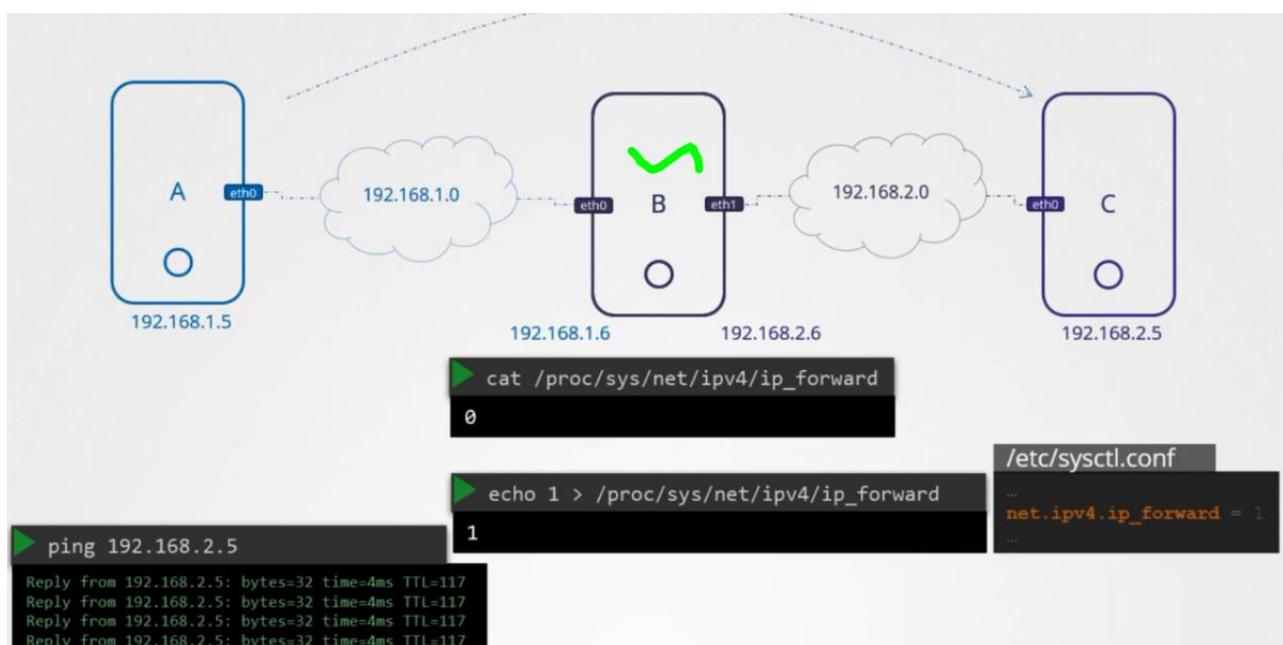
3 ➤ ping 192.168.1.11
Reply from 192.168.1.11: bytes=32 time=4ms TTL=117
Reply from 192.168.1.11: bytes=32 time=4ms TTL=117

Routers helps to connect two/many networks

Gateway – route to reach the destination



We need to forward B node eth0 to eth1.



Commands

```
ip link
```

```
ip addr
```

```
ip addr add 192.168.1.10/24 dev eth0
```

```
ip route
```

```
ip route add 192.168.1.0/24 via 192.168.2.1
```

```
cat /proc/sys/net/ipv4/ip_forward
```

```
1
```

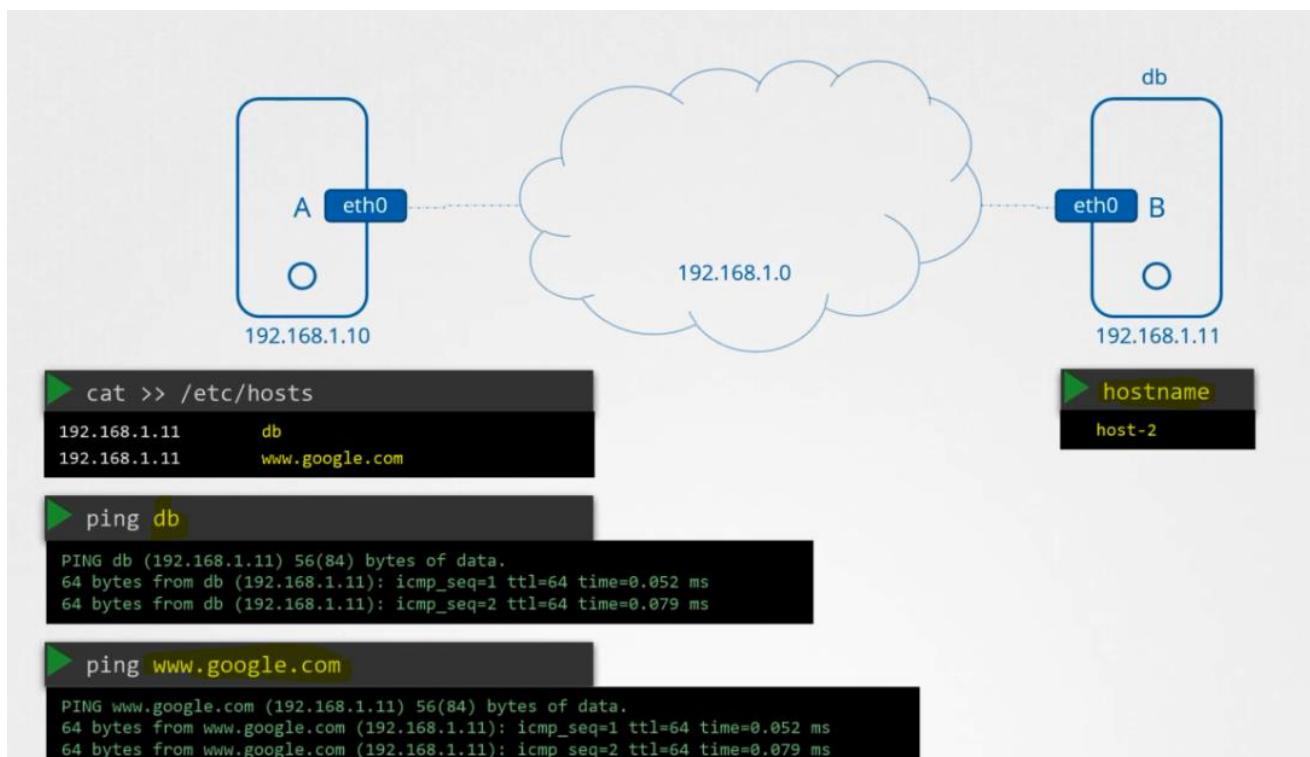
DNS

```
▶ ping db
ping: unknown host db

▶ cat >> /etc/hosts
192.168.1.11      db

▶ ping db
PING db (192.168.1.11) 56(84) bytes of data.
64 bytes from db (192.168.1.11): icmp_seq=1 ttl=64 time=0.052 ms
64 bytes from db (192.168.1.11): icmp_seq=2 ttl=64 time=0.079 ms
```

Host A always use its /etc/hosts file



By checking file -> getting ip is called Name resolutions

```

▶ cat >> /etc/hosts
192.168.1.11      db
192.168.1.11      www.google.com

▶ ping db

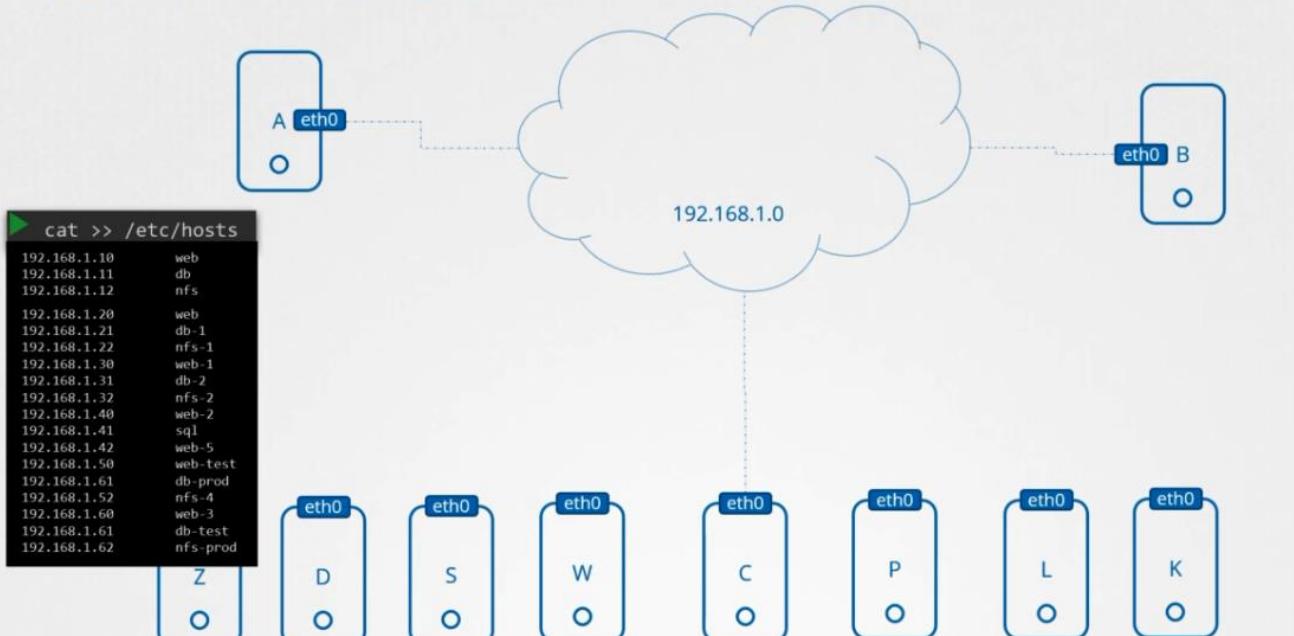
▶ ssh db

▶ curl http://www.google.com

```

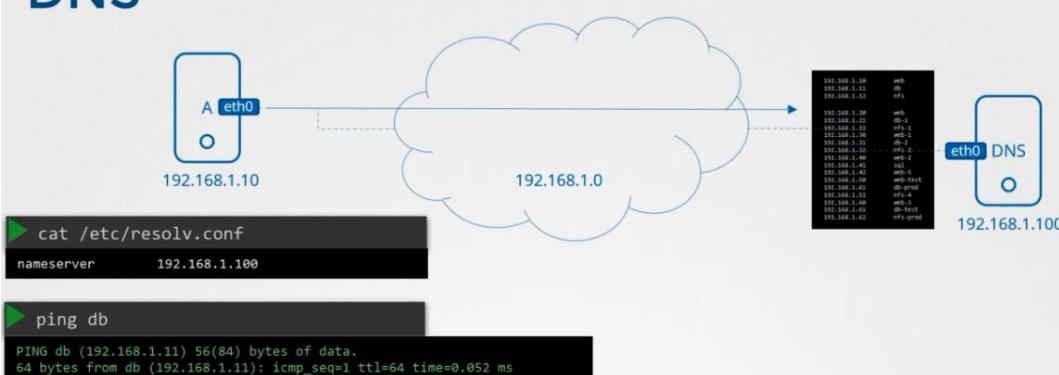
Keeping all the node names in /hosts file is very complex and everytime need to change if Ip is changing

Name Resolution

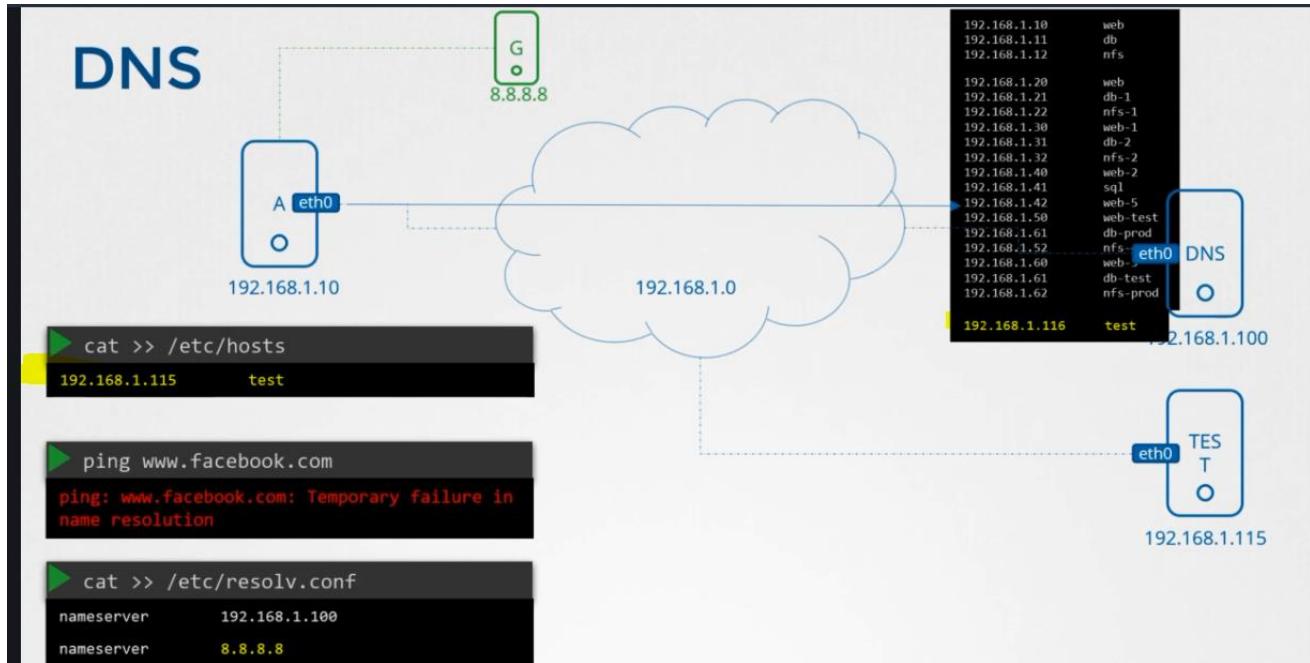


That's why we moved all the /hosts file to single server that's what called DNS server.all the hosts will point to the DNS server instead of its own /etc/hosts files.

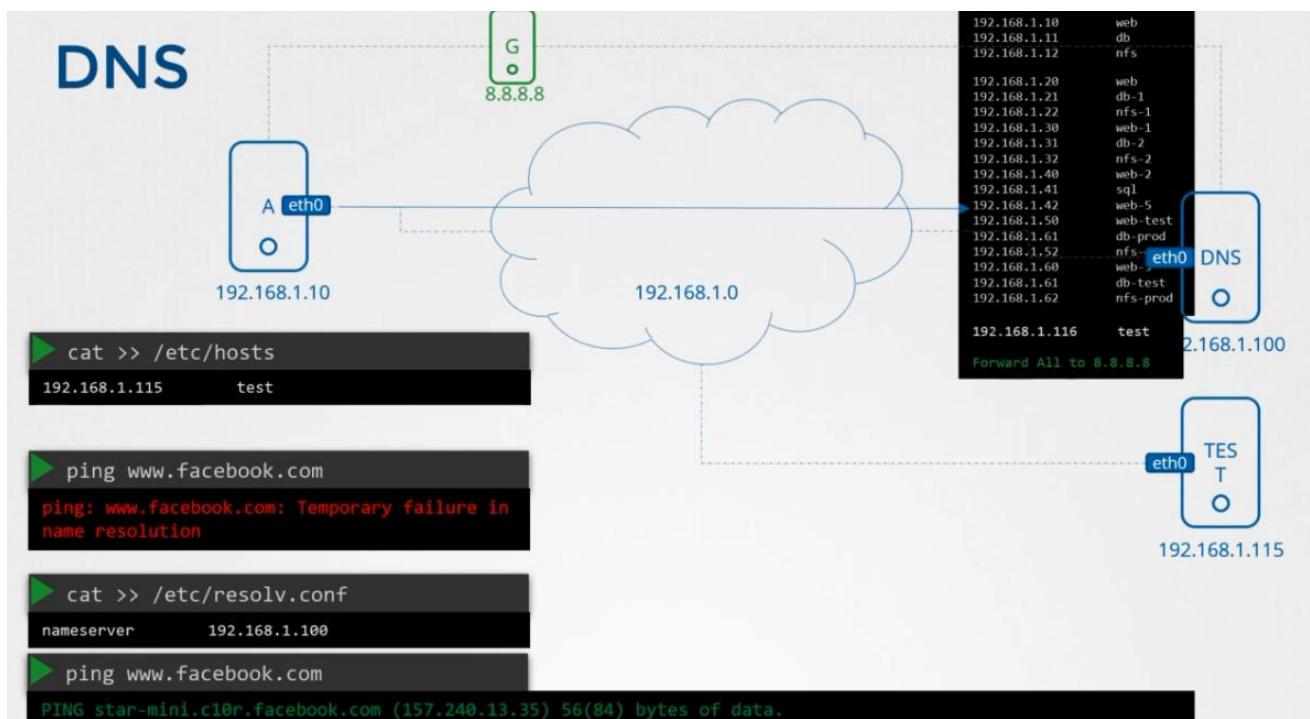
DNS



1st it looks for the local host file and then DNS server. Anyway this priority can be changed by editing the (xyz)file.



8.8.8.8 public name server from google

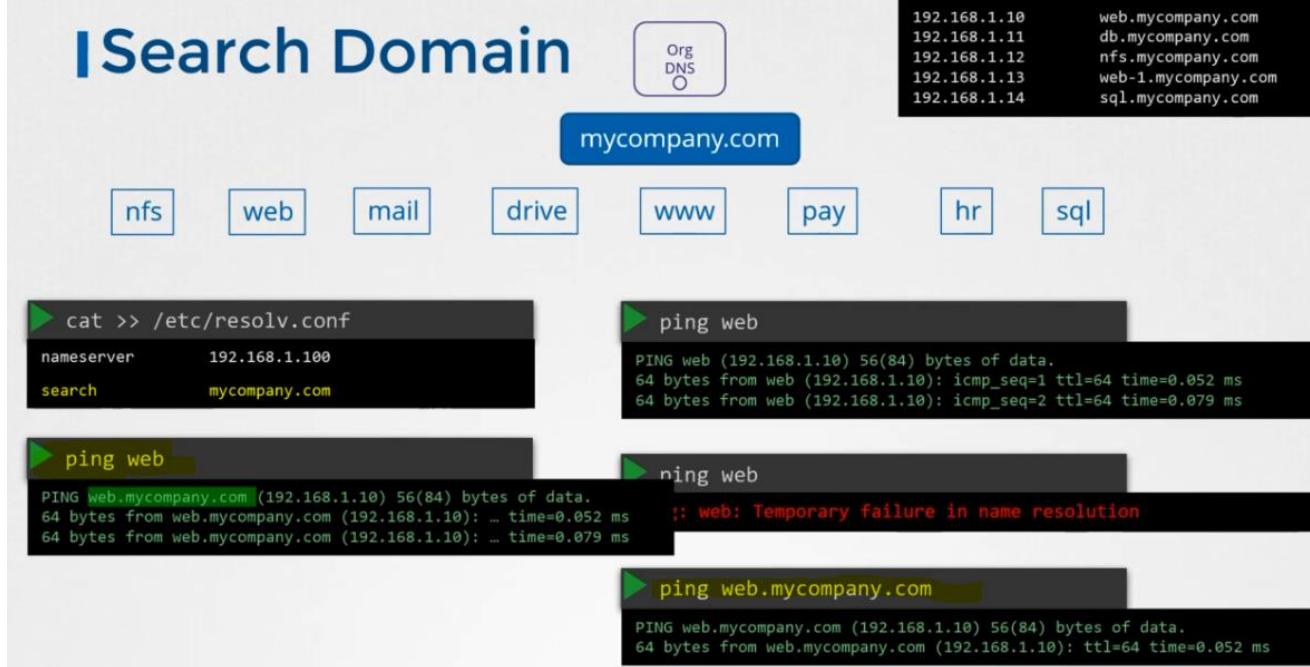


I Domain Names

apps.google.com



I Search Domain



Nslookup only check with DNS server

```
▶ nslookup www.google.com
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
Name:   www.google.com
Address: 172.217.0.132
```

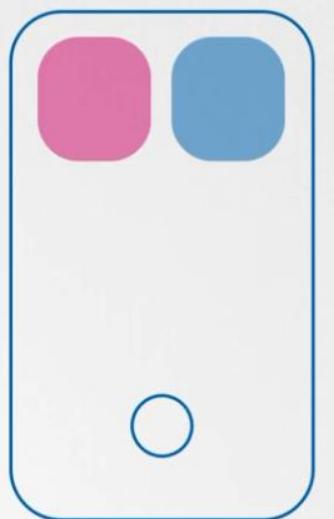
Network Namespaces are used by containers to create network isolation

NETWORK NAMESPACE



CREATE NETWORK NS

```
▶ ip netns add red
▶ ip netns add blue
▶ ip netns
red
blue
```



EXEC IN NETWORK NS

```
▶ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc state UP mode DEFAULT qlen 1000
    link/ether 02:42:ac:11:00:08 brd ff:ff:ff:ff:ff:ff
```

```
▶ ip netns exec red ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
▶ ip -n red link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
ip netns exec red ip link
=
ip -n red link
```

```
▶ arp
```

Address	Hwtype	Hwaddress	Flags Mask	Iface
172.17.0.21	ether	02:42:ac:11:00:15	C	eth0
172.16.0.8	ether	06:fe:d3:b5:59:65	C	eth0
_gateway	ether	02:42:d5:7a:84:8e	C	eth0
host01	ether	02:42:ac:11:00:1c	C	eth0

```
▶ ip netns exec red arp
```

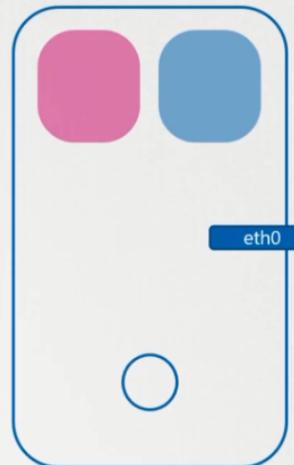
Address	Hwtype	Hwaddress	Flags Mask	Iface
---------	--------	-----------	------------	-------

```
▶ route
```

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
default	_gateway	0.0.0.0	UG	202	0	0	eth0
172.17.0.0	0.0.0.0	255.255.0.0	U	202	0	0	eth0
172.17.0.0	0.0.0.0	255.255.255.0	U	0	0	0	docker0

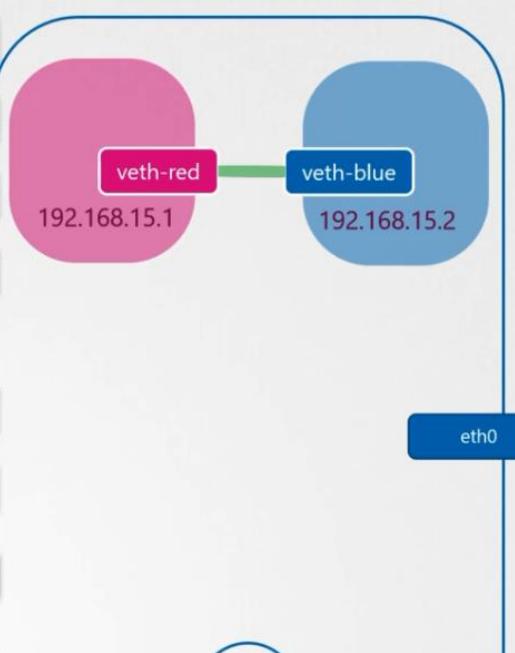
```
▶ ip netns exec red route
```

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface



Establish connection between namespaces

```
>ip link add veth-red type veth peer name veth-blue  
>ip link set veth-red netns red  
>ip link set veth-blue netns blue  
>ip -n red addr add 192.168.15.1 dev veth-red  
>ip -n blue addr add 192.168.15.2 dev veth-blue  
  
>ip -n red link set veth-red up  
>ip -n blue link set veth-blue up  
  
>ip netns exec red ping 192.168.15.2  
PING 192.168.15.2 (192.168.15.2) 56(84) bytes of data.  
64 bytes from 192.168.15.2: icmp_seq=1 ttl=64 time=0.026 ms
```



Separating namespace network from the host network

```
>ip netns exec red ping 192.168.15.2  
PING 192.168.15.2 (192.168.15.2) 56(84) bytes of data.  
64 bytes from 192.168.15.2: icmp_seq=1 ttl=64 time=0.026 ms  
  
>ip netns exec red arp  
Address Hwtype Hwaddress Flags Mask Iface  
192.168.15.2 ether ba:b0:6d:68:09:e9 C veth-red  
  
>ip netns exec blue arp  
Address Hwtype Hwaddress Flags Mask Iface  
192.168.15.1 ether 7a:9d:9b:c8:3b:7f C veth-blue  
  
>arp  
Address Hwtype Hwaddress Flags Mask Iface  
192.168.1.3 ether 52:54:00:12:35:03 C eth0  
192.168.1.4 ether 52:54:00:12:35:04 C eth0
```

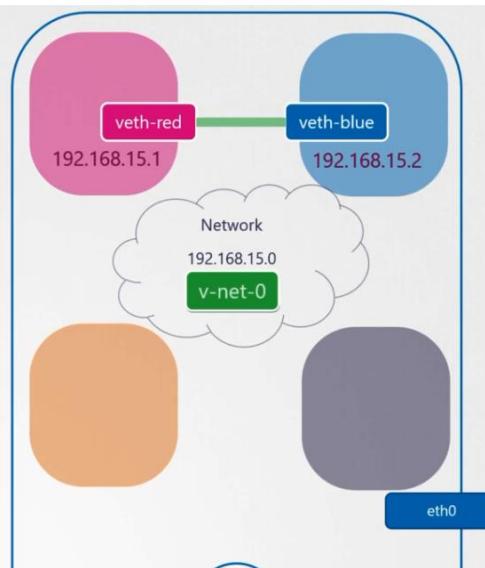
When multiple namespaces are there, we have to create virtual switch/bridge and connect them

LINUX BRIDGE

```
▶ ip link add v-net-0 type bridge
```

```
▶ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 02:0d:31:14:c7:a7 brd ff:ff:ff:ff:ff:ff
6: v-net-0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
mode DEFAULT group default qlen 1000
    link/ether 06:9d:69:52:6f:61 brd ff:ff:ff:ff:ff:ff
```

```
▶ ip link set dev v-net-0 up
```

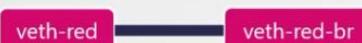


Now we'll delete the previous link.

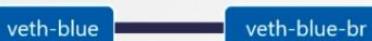
```
▶ ip -n red link del veth-red
```

1. create the virtual cables

```
▶ ip link add veth-red type veth peer name veth-red-br
```



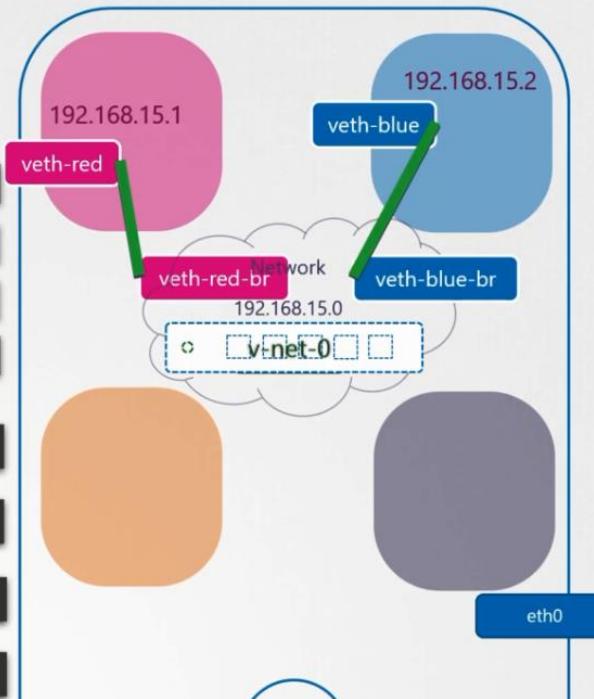
```
▶ ip link add veth-blue type veth peer name veth-blue-br
```



2. Now attach the created cable to the vSwitch by following command and assign IP address. So that it can ping each other through vNetwork

LINUX BRIDGE

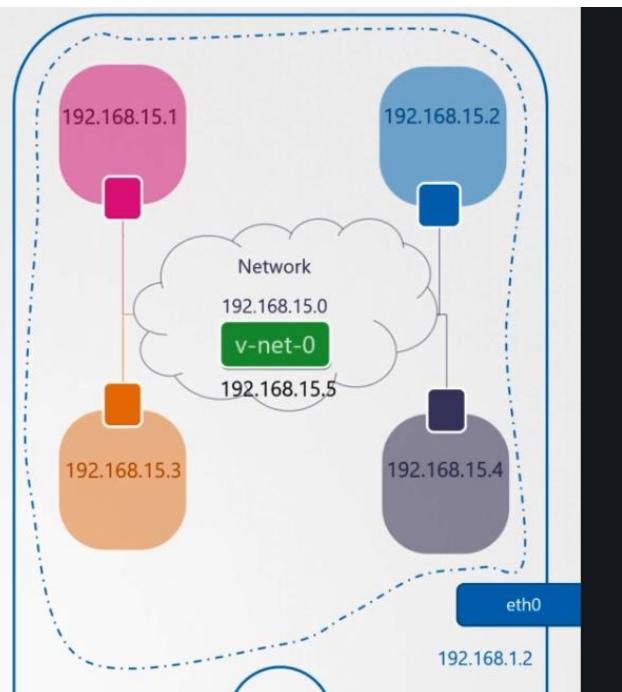
```
▶ ip link set veth-red netns red  
▶ ip link set veth-red-br master v-net-0  
▶ ip link set veth-blue netns blue  
▶ ip link set veth-blue-br master v-net-0  
  
▶ ip -n red addr add 192.168.15.1 dev veth-red  
▶ ip -n blue addr add 192.168.15.2 dev veth-blue  
  
▶ ip -n red link set veth-red up  
▶ ip -n blue link set veth-blue up
```

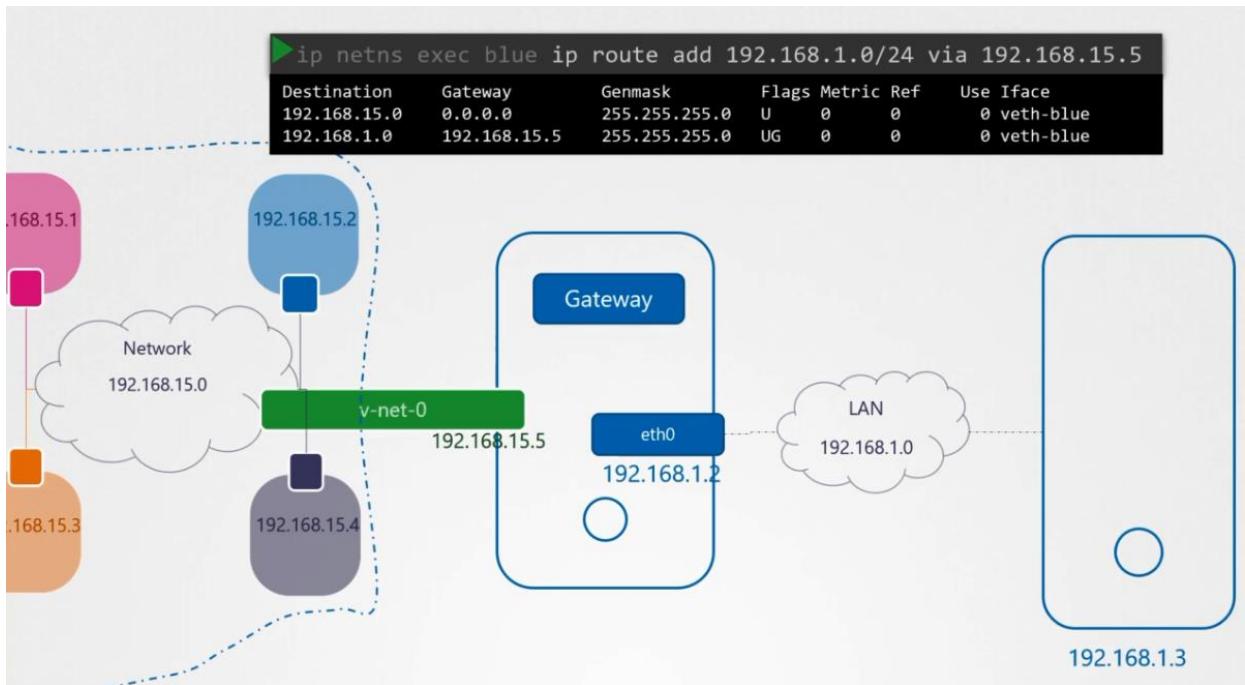


Private vNetwork

LINUX BRIDGE

```
▶ ping 192.168.15.1  
Not Reachable!  
  
▶ ip addr add 192.168.15.5/24 dev v-net-0  
  
▶ ping 192.168.15.1  
PING 192.168.15.1 (192.168.15.1) 56(84) bytes of data.  
64 bytes from 192.168.15.1: icmp_seq=1 ttl=64 time=0.026 ms
```





Add NAT to forward the request. Now able to ping from vNetwork to outside localNetwork

```

iptables -t nat -A POSTROUTING -s 192.168.15.0/24 -j MASQUERADE

ip netns exec blue ping 192.168.1.3
64 bytes from 192.168.1.3: icmp_seq=1 ttl=63 time=0.587 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=63 time=0.466 ms

```

vNetwork to outside world Internet

```

ip netns exec blue ping 8.8.8.8
Connect: Network is unreachable

ip netns exec blue route
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
192.168.15.0    0.0.0.0        255.255.255.0 U       0      0      0 veth-blue
192.168.1.0     192.168.15.5   255.255.255.0 UG      0      0      0 veth-blue

ip netns exec blue ip route add default via 192.168.15.5
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
192.168.15.0    0.0.0.0        255.255.255.0 U       0      0      0 veth-blue
192.168.1.0     192.168.15.5   255.255.255.0 UG      0      0      0 veth-blue
Default         192.168.15.5   255.255.255.0 UG      0      0      0 veth-blue

ip netns exec blue ping 8.8.8.8

```

Outside world to vNetwork not able to connect. We need to enable port forwarding rule

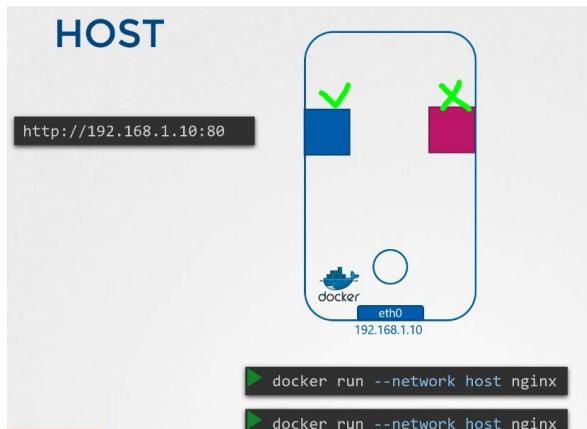
Docker Network

1. Docker with no network
2. Docker with host network
3. Docker with Bridge network

➤ Docker with no network

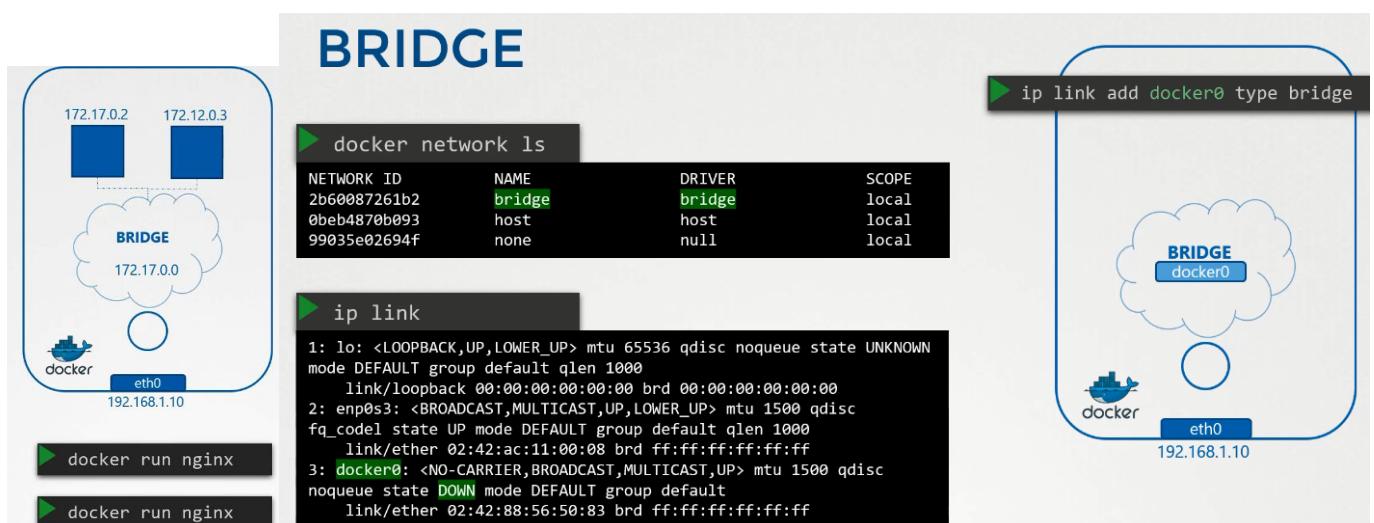
```
▶ docker run --network none nginx
```

➤ Docker with Host



Only one container can use host port. Two process cannot listen on the same port

➤ Docker with Bridge



Docker creates network namespace when we run the docker run command.

Container == network namespace.

Docker also follows the same strategy like Network namespace-Linux bridge.

BRIDGE

```
▶ ip netns
```

```
b3165c10a92b
```

```
▶ ip link
```

```
...
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default
    link/ether 02:42:b9:5f:d6:21 brd ff:ff:ff:ff:ff:ff
8: vethbb1c343@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0
state UP mode DEFAULT group default
    link/ether 9e:71:37:83:9f:50 brd ff:ff:ff:ff:ff:ff link-netnsid 1
```

```
▶ ip -n b3165c10a92b link
```

```
...
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

BRIDGE docker0

172.17.0.1

192.168.1.10

```
▶ docker run nginx
```

```
2e41deb9ef1b8b3d141c7bb5d883541b4d56c21cf055e236f87
```

Created vLink and attached to the bridge and container. Container also gets Ip address

```
▶ ip netns
```

```
b3165c10a92b
```

```
▶ ip -n b3165c10a92b addr
```

```
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
        valid_lft forever preferred_lft forever
```

Procedure after all docker containers

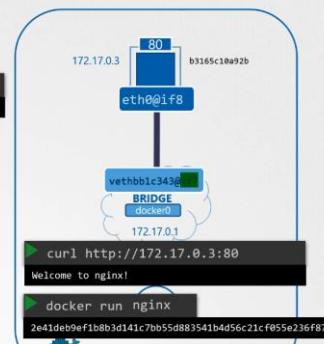
1. Docker creates the namespace
2. Create pair of interfaces (links)
3. Attach one end to the container
4. And another end to the bridge network

BRIDGE

```
▶ curl http://172.17.0.3:80
```

```
curl: (7) Failed to connect... No route to host
```

outside world
Cannot talk
with Docker



```
▶ curl http://172.17.0.3:80
```

```
Welcome to nginx!
```

```
▶ docker run nginx
```

```
2e41deb9ef1b8b3d141c7bb5d883541b4d56c21cf055e236f87
```

To access from outside to docker , it provides Port Mapping

or Forwarding option. ➤ docker run -p 8080:80 nginx

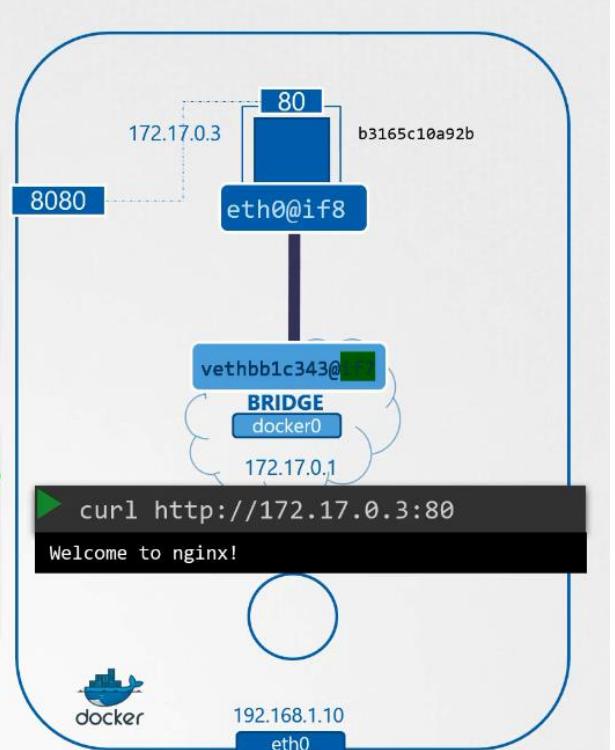
```
➤ curl http://192.168.1.10:8080  
Welcome to nginx!
```

How Docker is doing port forwarding ?

BRIDGE

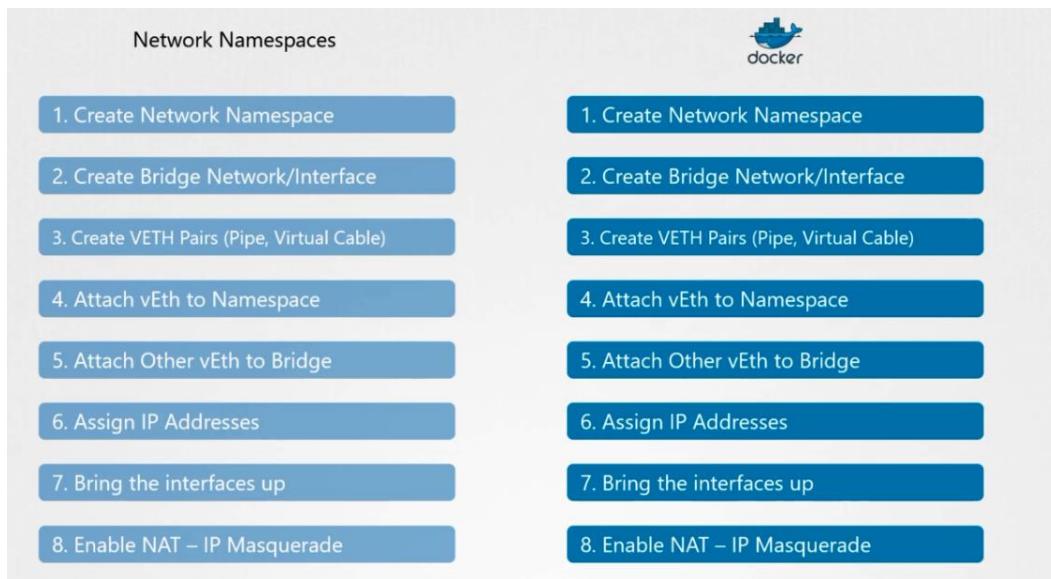
```
➤ iptables \Host way \  
-t nat \  
-A PREROUTING \  
-j DNAT \  
--dport 8080 \  
--to-destination 80
```

```
➤ iptables \Docker way \  
-t nat \  
-A DOCKER \  
-j DNAT \  
--dport 8080 \  
--to-destination 172.17.0.3:80
```

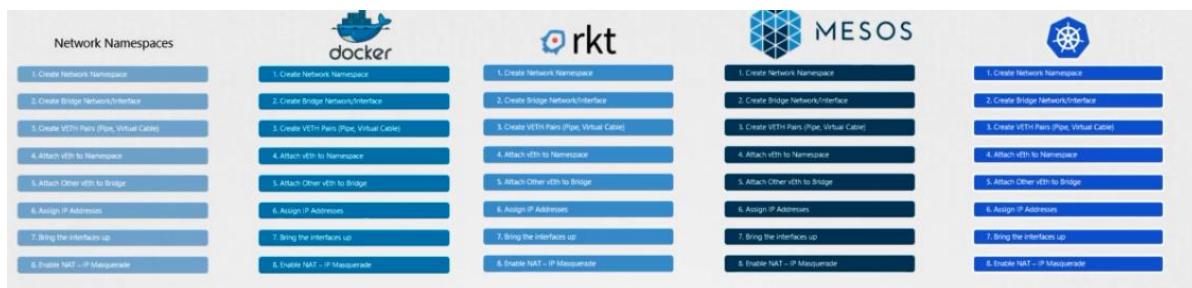


Check the rules

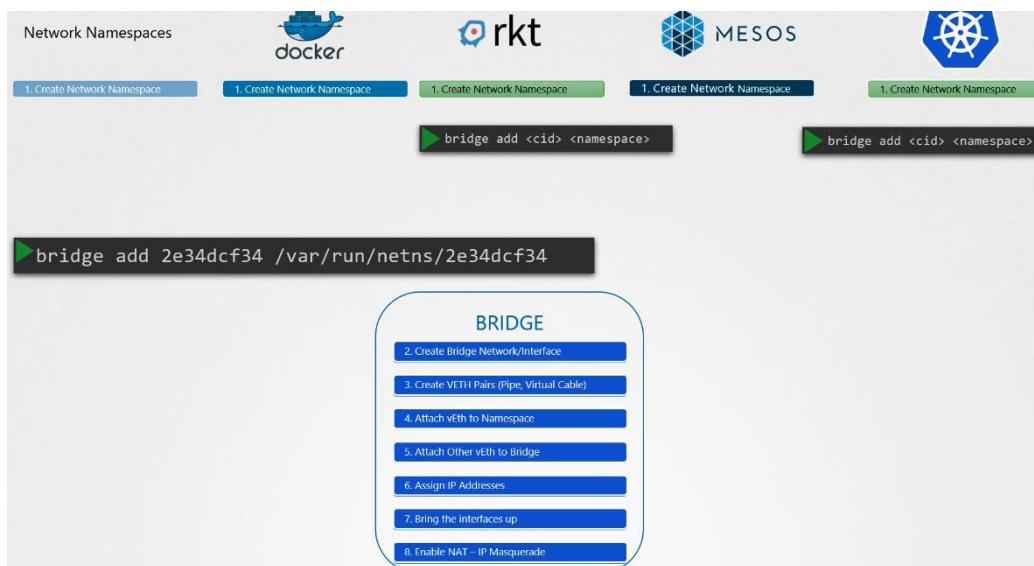
```
➤ iptables -nvL -t nat  
Chain DOCKER (2 references)  
target      prot opt source          destination  
RETURN     all   --  anywhere       anywhere  
DNAT       tcp   --  anywhere       anywhere           tcp dpt:8080 to:172.17.0.2:80
```



Same approach in all , so why repeated steps ? . better make it as a single standard,use it everywhere



Create Bridge (script), so any one can reuse it.





CONTAINER NETWORK INTERFACE

- Container Runtime must create network namespace
- Identify network the container must attach to
- Container Runtime to invoke Network Plugin (bridge) when container is ADDED.
- Container Runtime to invoke Network Plugin (bridge) when container is DELETED.
- JSON format of the Network Configuration

rkt



BRIDGE

1. Create Bridge Network/Interface
2. Create VETH Pairs (Pipe, Virtual Cable)
3. Attach veth to Namespace
4. Attach Other veth to Bridge
5. Assign IP Addresses
6. Bring the interfaces up
7. Enable NAT – IP Masquerade

The below all are implements the CNI standard, except Docker



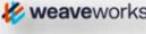
CONTAINER NETWORK INTERFACE

rkt

MESOS


BRIDGE
VLAN
IPVLAN
MACVLAN
WINDOWS

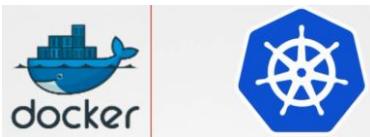
DHCP
host-local

 weaveworks
 flannel
 cilium
 NSX


Docker has its own standard called CNM



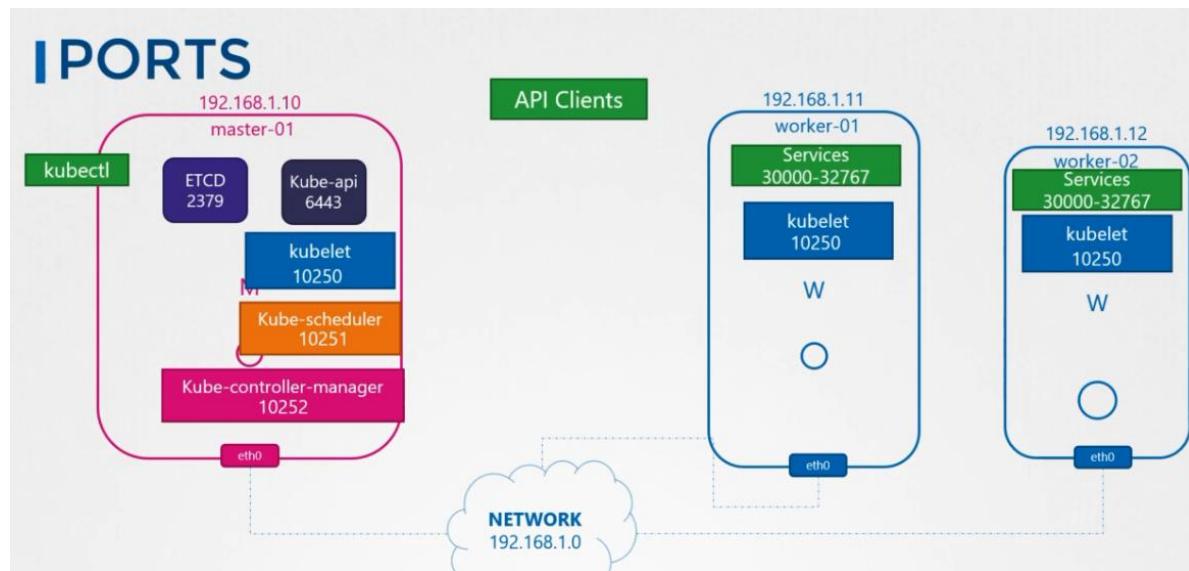
CONTAINER NETWORK MODEL (CNM)



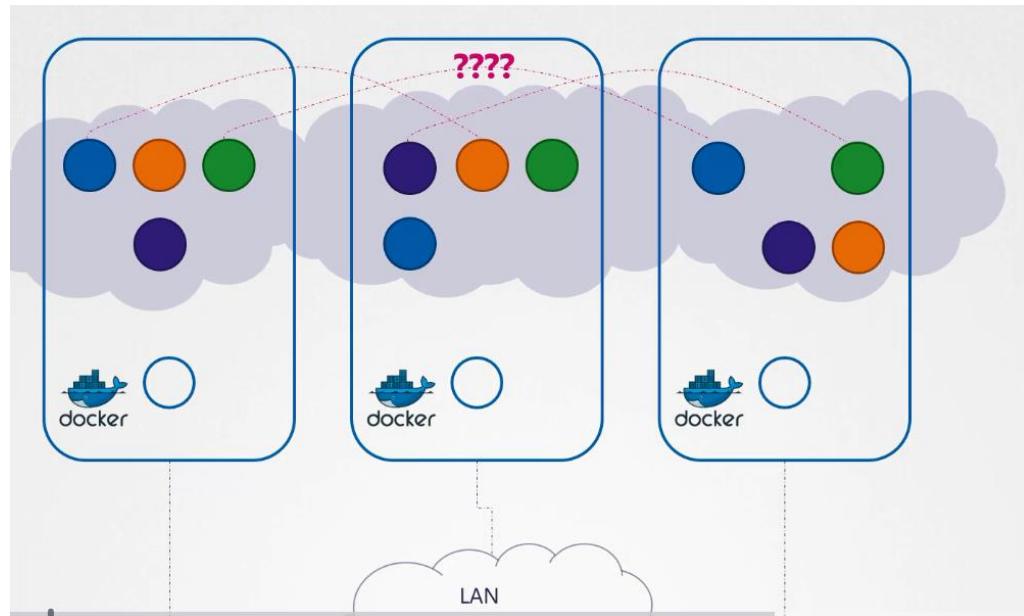
```
▶ docker run --network=none nginx
```

```
▶ bridge add 2e34dcf34 /var/run/netns/2e34dcf34
```

Create docker network with none and manually add the CNI standard N/w

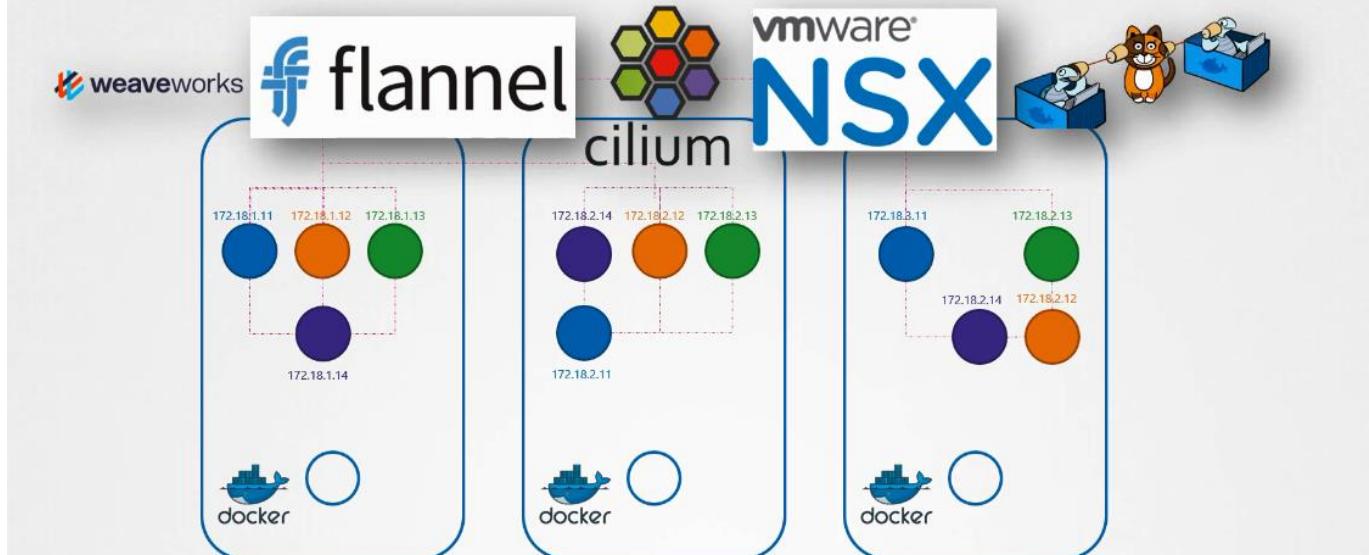


Pod networking : k8s does not come with readymade solution for this



Networking Model

- ❑ Every POD should have an IP Address
- ❑ Every POD should be able to communicate with every other POD in the same node.
- ❑ Every POD should be able to communicate with every other POD on other nodes without NAT.



Mauly solve N/w solution

1. We have 3 nodes, all assigned IP
2. When containers are created, k8s create namespace for them
3. To enable communicate between them we attach these namespaces to the network. Like in Linux – Bridge Network
4. Create Bridge network in each node and bring them up
5. Assign Ip to the Bridge Network . we decide each bridge n/w has its own subnet
6. Assign IP address for the Bridge Interface

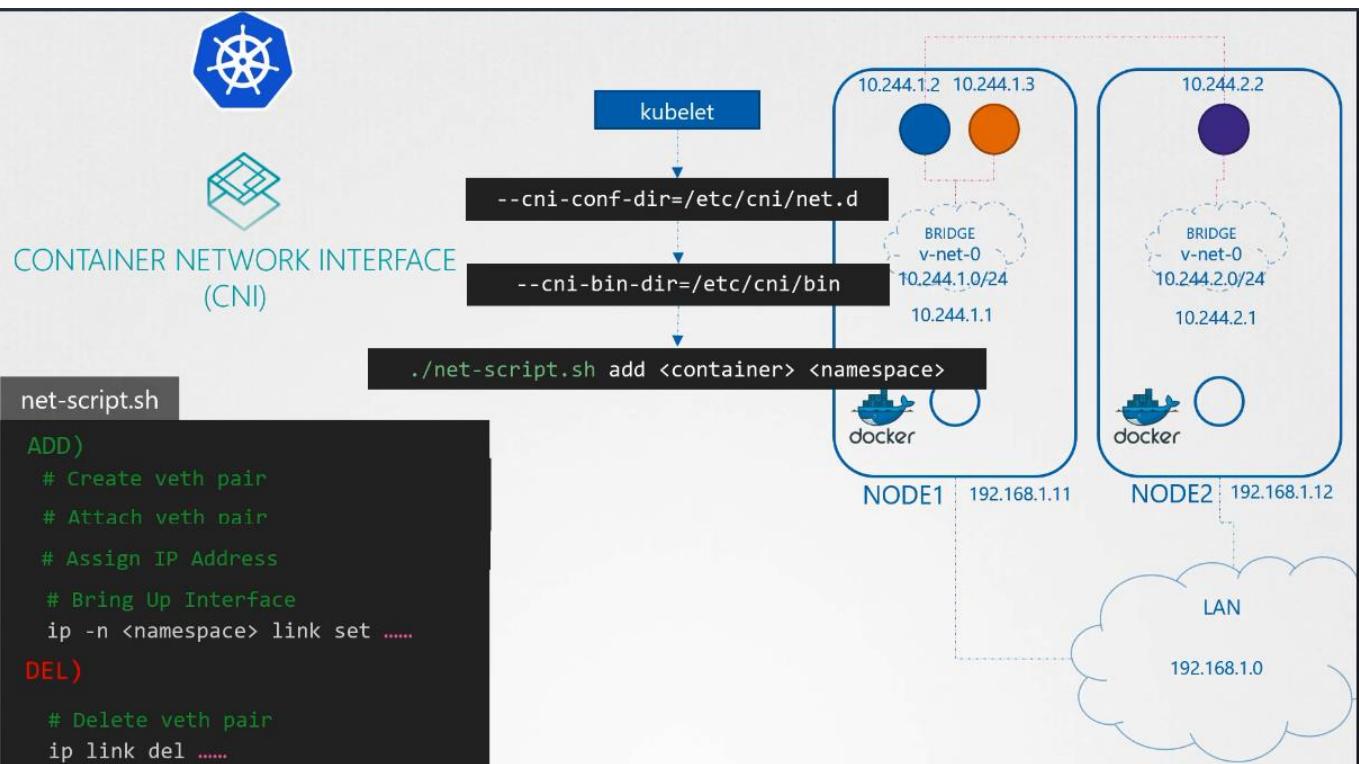
Bridge script

7. Create a wire/link
8. Attach the wire one end at the container another end at the bridge
9. Assign the IP and add the default route – gateway
10. Bring up the interface

Enable communication between one pod to another pod that is in another Node

11. Add IP route -> via node 2 . then pod from node1 can be communicate with pod in node2
12. Add IP route in all the node.

When kubelet creates the container , it looks for the script and runs the script



CNI is set to the kubelet services

Configuring CNI

kubelet.service

```

ExecStart=/usr/local/bin/kubelet \
--config=/var/lib/kubelet/kubelet-config.yaml \
--container-runtime=remote \
--container-runtime-endpoint=unix:///var/run/containerd/containerd.sock \
--image-pull-progress-deadline=2m \
--kubeconfig=/var/lib/kubelet/kubeconfig \
--network-plugin=cni \
--cni-bin-dir=/opt/cni/bin \
--cni-conf-dir=/etc/cni/net.d \
--register-node=true \
--v=2

```

```
ps -aux | grep kubelet
```

```

root      2095  1.8  2.4 960676 98788 ?          Ssl  02:32  0:36 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/lib/kubelet/config.yaml --cgroup-driver=cgroupfs --cni-bin-dir=/opt/cni/bin --cni-conf-dir=/etc/cni/net.d --network-plugin=cni

```

I View kubelet options

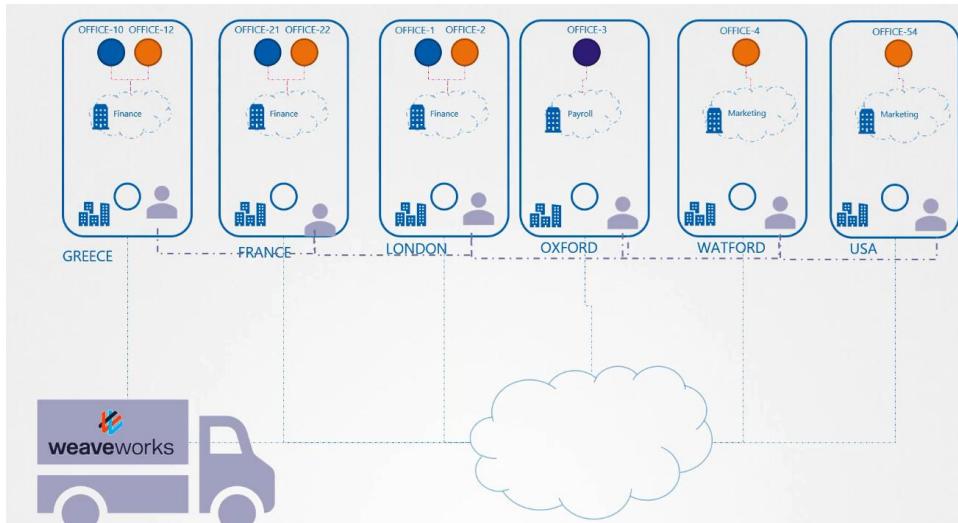
```
▶ ls /etc/cni/net.d  
10-bridge.conf
```



```
▶ cat /etc/cni/net.d/10-bridge.conf
```

```
{  
    "cniVersion": "0.2.0",  
    "name": "mynet",  
    "type": "bridge",  
    "bridge": "cni0",  
    "isGateway": true,  
    "ipMasq": true,  
    "ipam": {  
        "type": "host-local",  
        "subnet": "10.22.0.0/16",  
        "routes": [  
            { "dst": "0.0.0.0/0" }  
        ]  
    }  
}
```

Weave plugin



IPAM – Ip management

kube-proxy

iptables

IP : Port	Forward To:
10.99.13.178:80	10.244.1.2

```
▶ kubelet get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
db	1/1	Running	0	14h	10.244.1.2	node-1

```
▶ kubelet get service
```

NAME	TYPE	CLUSTER-IP	PORT(S)	AGE
db-service	ClusterIP	10.103.132.104	3306/TCP	12h

```
▶ iptables -L -t net | grep db-service
```

KUBE-SVC-XA50GUC7YRHOS3PU	tcp	--	anywhere	10.103.132.104	/* defau
DNAT	tcp	--	anywhere	anywhere	/* defau
KUBE-SEP-JBWCWHHQM57V2WN7	all	--	anywhere	anywhere	/* defau