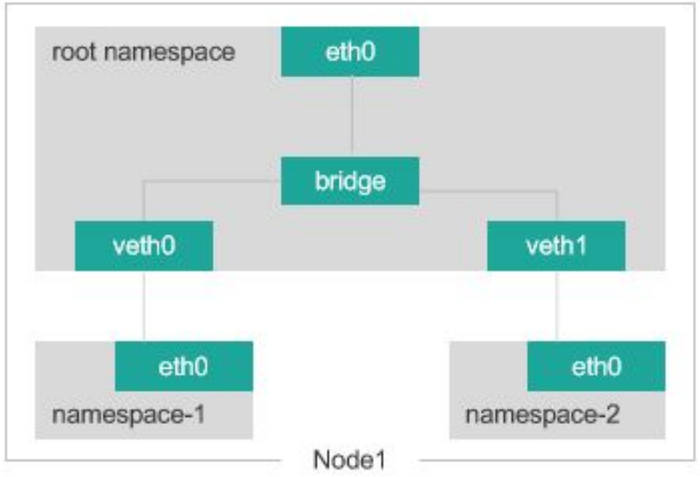
Networking with Kubernetes!

Part 1

Intro: K8S Networking

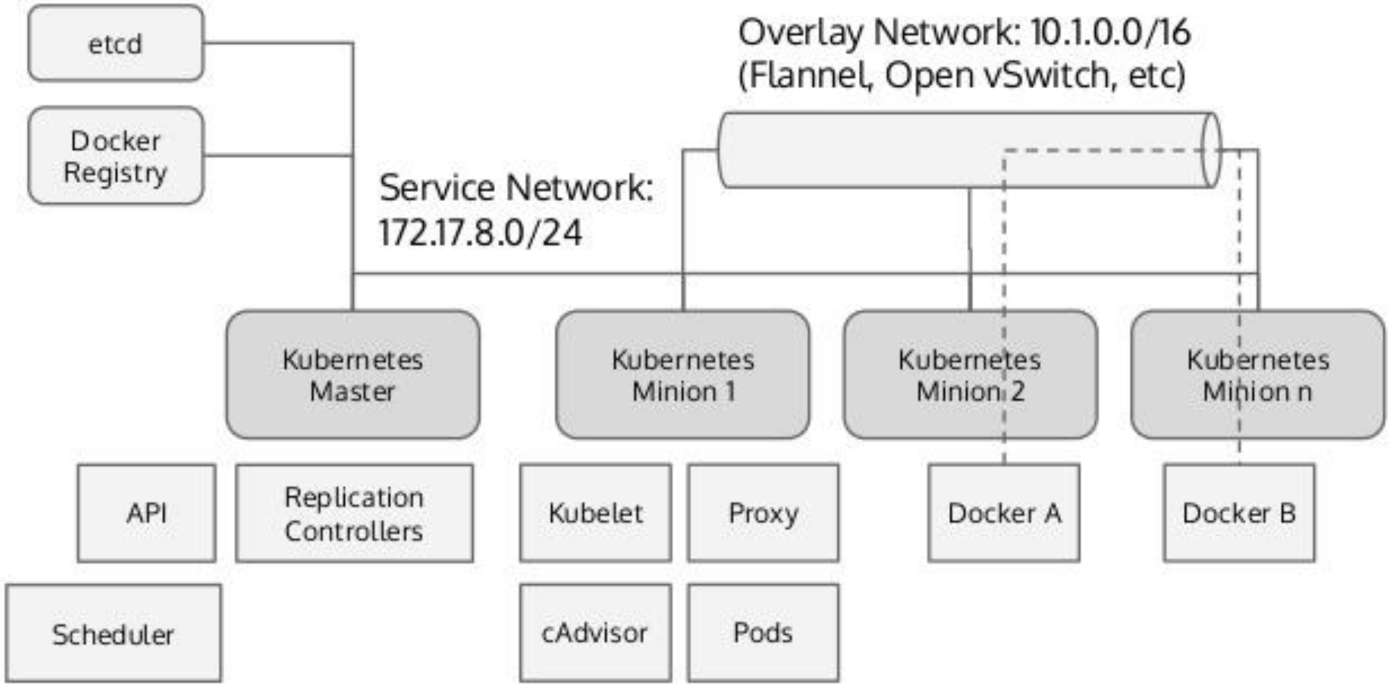
Namespace

- Linux kernel has 6 types of namespaces: *pid,net,mnt,uts,ipc,user*
- Network namespaces provide a brand-new network stack for all the processes within the namespace.
- That includes network interfaces,routing tables and iptables rules

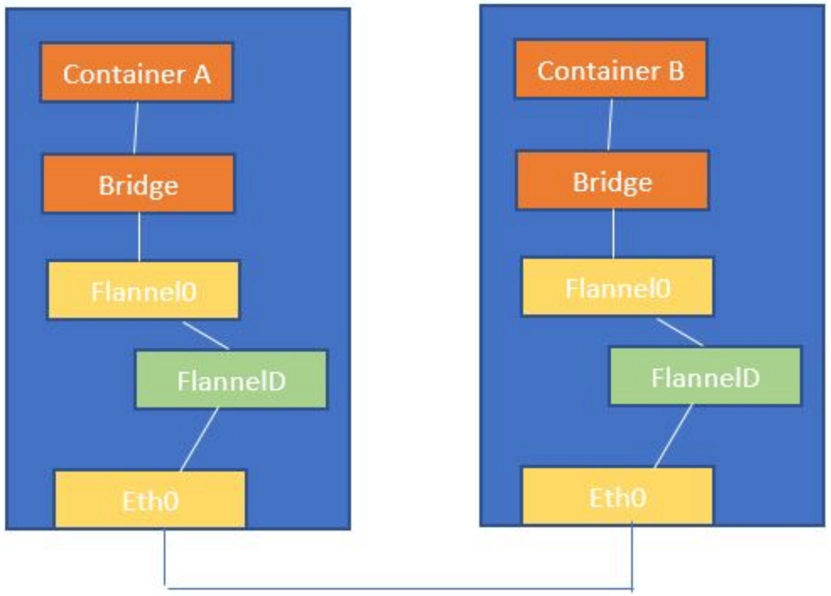


Live demo: Play with namespaces and cgroups.

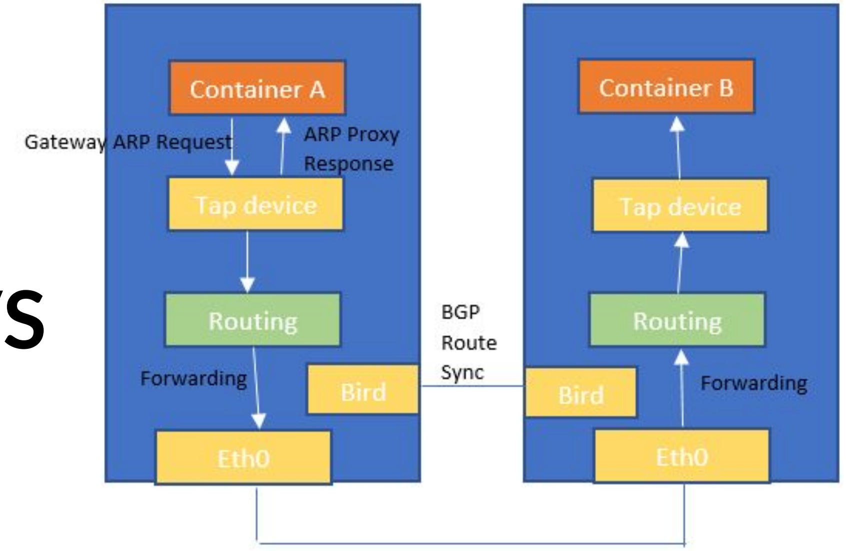
Kubernetes Architecture



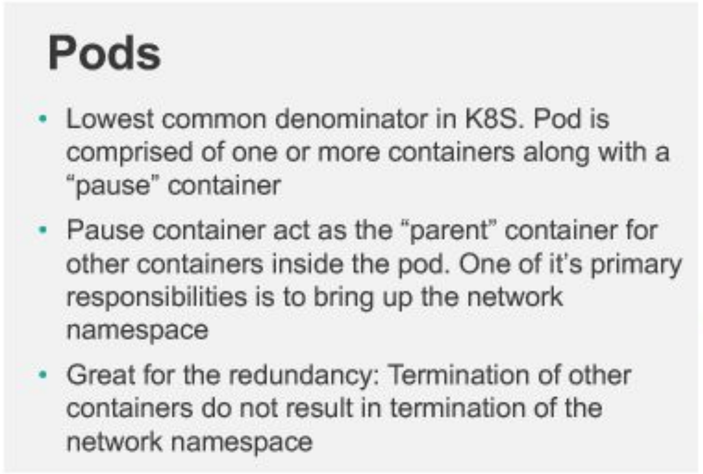
Flannel VS Calico



VS

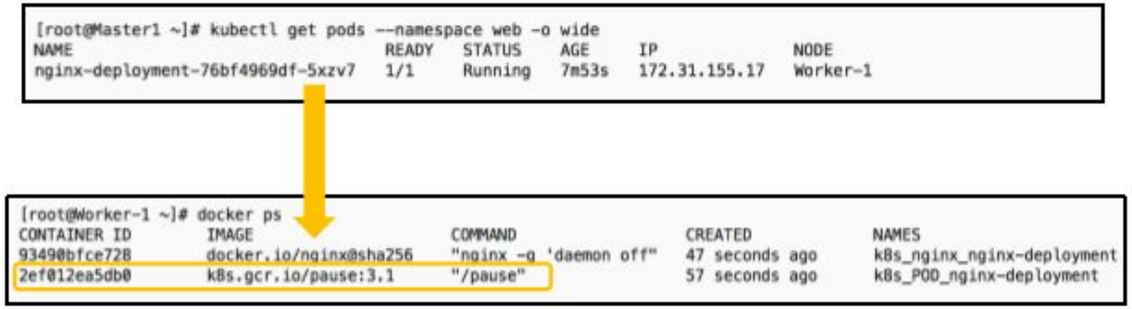


Pods



Pods

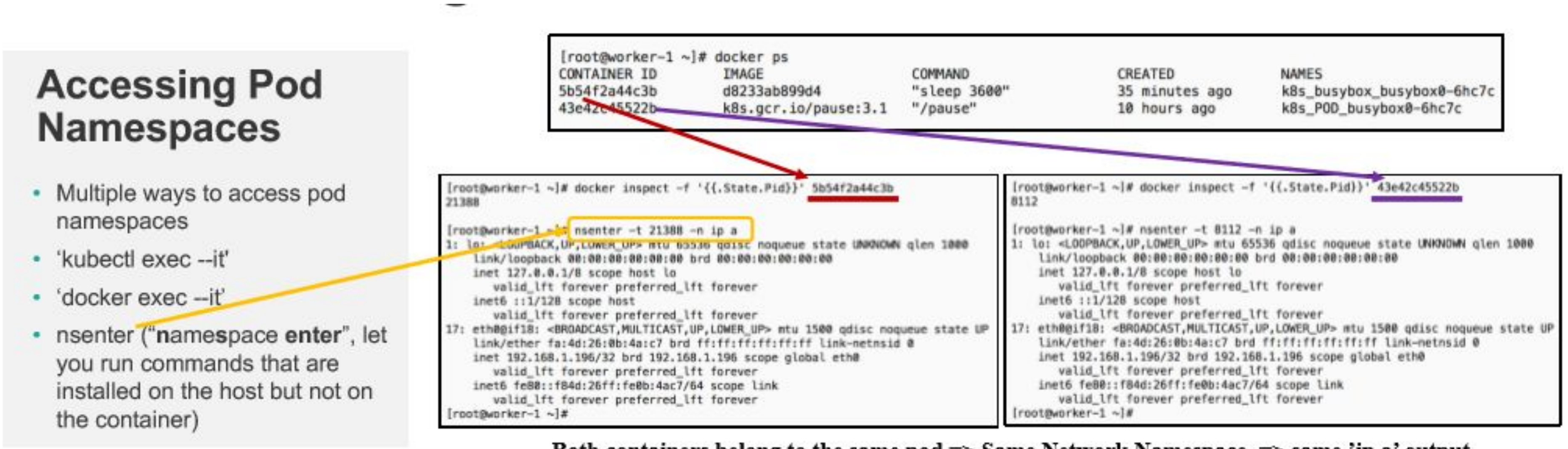
- Lowest common denominator in K8S. Pod is comprised of one or more containers along with a “pause” container
- Pause container act as the “parent” container for other containers inside the pod. One of it's primary responsibilities is to bring up the network namespace
- Great for the redundancy: Termination of other containers do not result in termination of the network namespace



```
[root@Master1 ~]# kubectl get pods --namespace web -o wide
NAME                              READY   STATUS    AGE     IP              NODE
nginx-deployment-76bf4969df-5xzv7   1/1     Running   7m53s   172.31.155.17   Worker-1
```

```
[root@Worker-1 ~]# docker ps
CONTAINER ID        IMAGE                       COMMAND                    CREATED             NAMES
93490bfce728        docker.io/nginx@sha256      "nginx -g 'daemon off"     47 seconds ago      k8s_nginx_nginx-deployment
2ef012ea5db0        k8s.gcr.io/pause:3.1        "/pause"                   57 seconds ago      k8s_POD_nginx-deployment
```

Accessing Pod Namespaces



Accessing Pod Namespaces

- Multiple ways to access pod namespaces
- ‘kubectl exec --it’
- ‘docker exec --it’
- nsenter (“**name**space **enter**”, let you run commands that are installed on the host but not on the container)

```
[root@worker-1 ~]# docker ps
CONTAINER ID        IMAGE                  COMMAND              CREATED            NAMES
5b54f2a44c3b        d8233ab899d4           "sleep 3600"         35 minutes ago     k8s_busybox_busybox0-6hc7c
43e42c45522b        k8s.gcr.io/pause:3.1   "/pause"             10 hours ago       k8s_POD_busybox0-6hc7c
```

```
[root@worker-1 ~]# docker inspect -f '{{.State.Pid}}' 5b54f2a44c3b
21388

[root@worker-1 ~]# nsenter -t 21388 -n ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
17: eth0@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether fa:4d:26:0b:4a:c7 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.196/32 brd 192.168.1.196 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::f84d:26ff:fe0b:4ac7/64 scope link
       valid_lft forever preferred_lft forever
[root@worker-1 ~]#
```

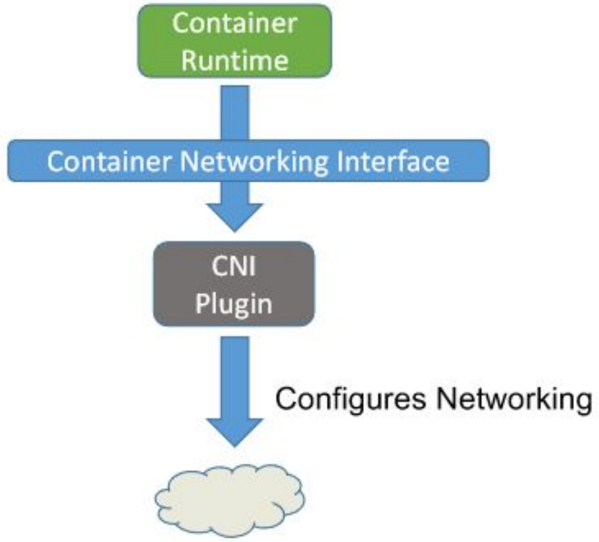
```
[root@worker-1 ~]# docker inspect -f '{{.State.Pid}}' 43e42c45522b
8112

[root@worker-1 ~]# nsenter -t 8112 -n ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
17: eth0@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether fa:4d:26:0b:4a:c7 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.196/32 brd 192.168.1.196 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::f84d:26ff:fe0b:4ac7/64 scope link
       valid_lft forever preferred_lft forever
[root@worker-1 ~]#
```

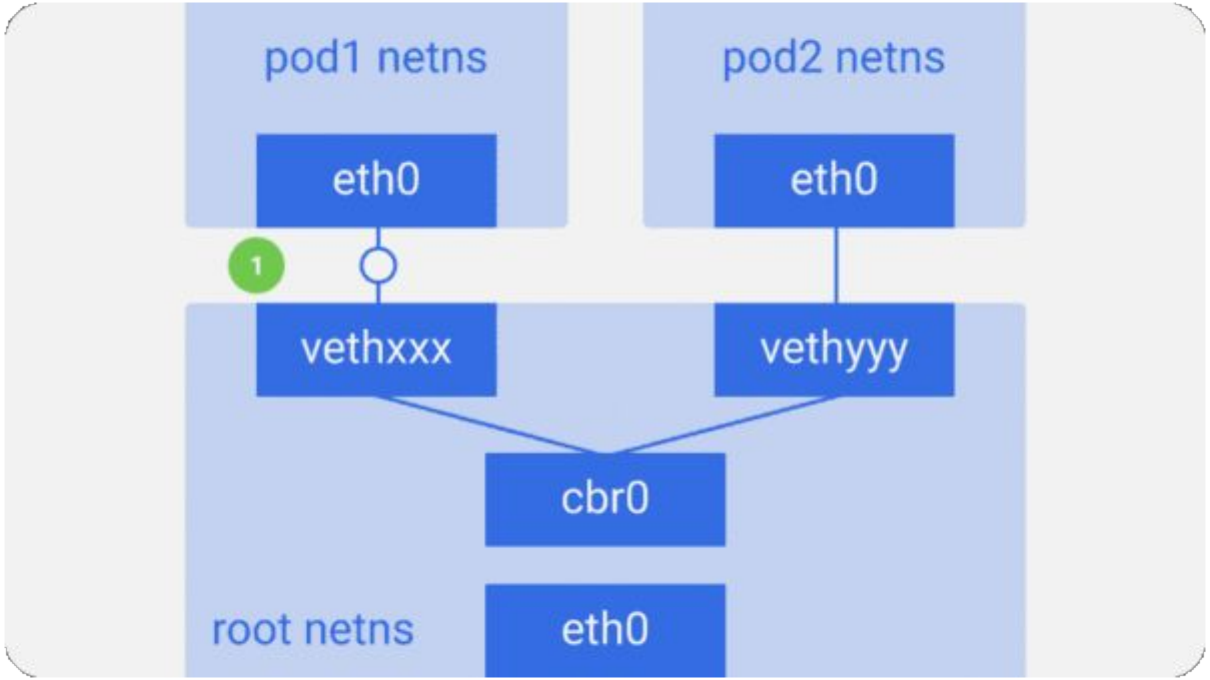
Both containers belong to the same pod => Same Network Namespace => same ’ip a’ output

Container Networking Interface : CNI

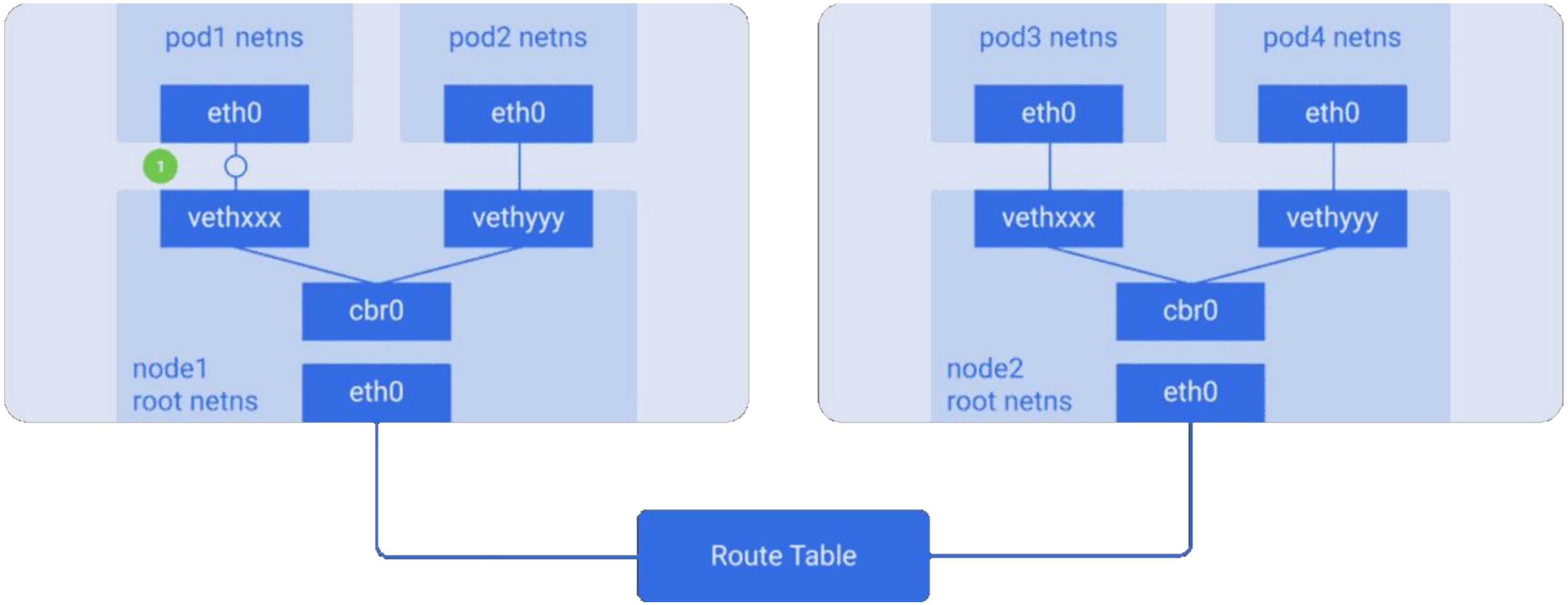
- Interface between container runtime and network implementation
- Network plugin implements the CNI spec. It takes a container runtime and configure (attach/detach) it to the network
- CNI plugin is an executable (in: /opt/cni/bin)
- When invoked it reads in a JSON config & Environment Variables to get all the required parameters to configure the container with the network



Intra-node communication



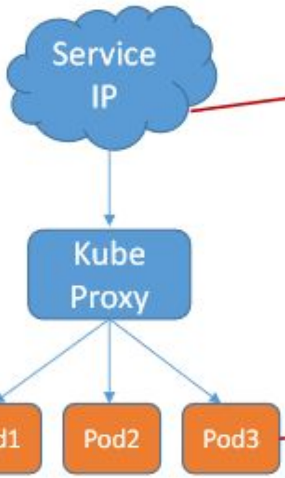
Inter-node communication

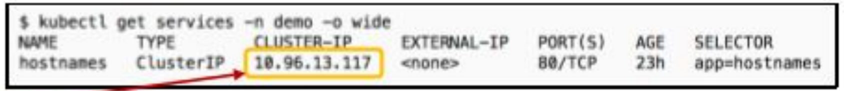


Back to the Basics

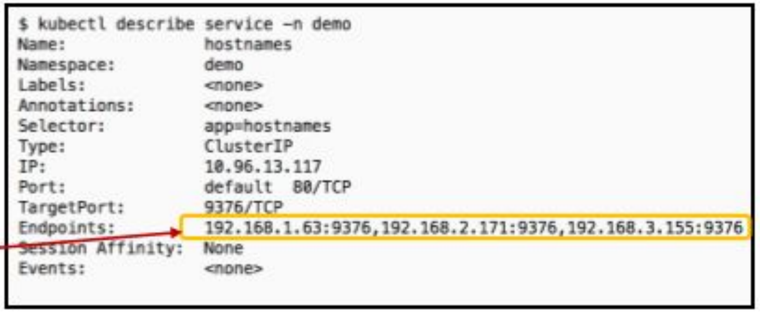
Services

- Pods are mortal
- Need a higher level abstractions: Services
- "Service" in Kubernetes is a conceptual concept. Service is not a process/daemon. Outside networks doesn't learn Service IP addresses
- Implemented through Kube Proxy with IPTables rules



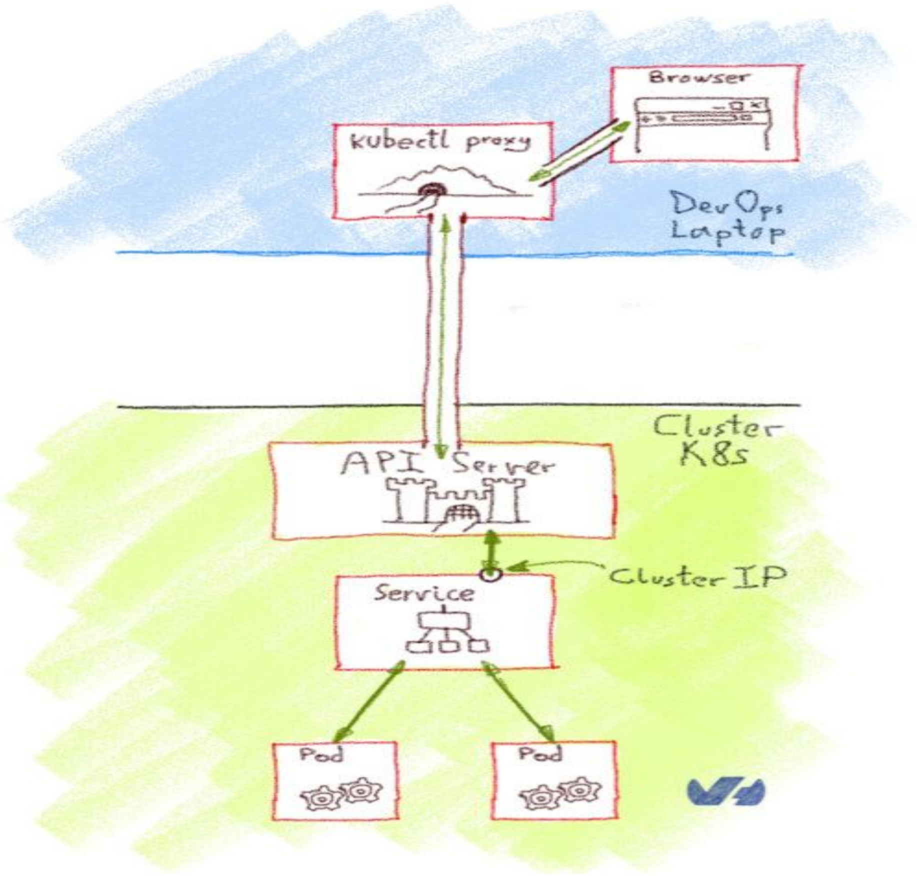


```
$ kubectl get services -n demo -o wide
NAME        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE   SELECTOR
hostnames   ClusterIP   10.96.13.117   <none>        80/TCP    23h   app=hostnames
```

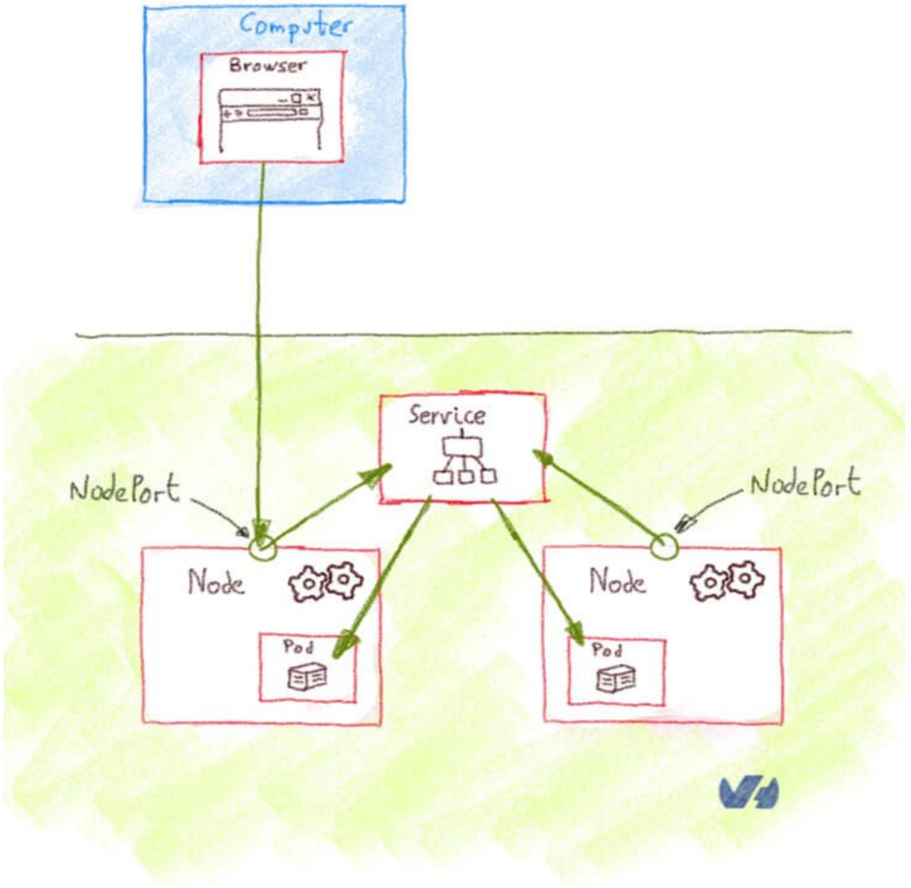


```
$ kubectl describe service -n demo
Name:              hostnames
Namespace:         demo
Labels:            <none>
Annotations:       <none>
Selector:          app=hostnames
Type:              ClusterIP
IP:                10.96.13.117
Port:              default  80/TCP
TargetPort:        9376/TCP
Endpoints:         192.168.1.63:9376,192.168.2.171:9376,192.168.3.155:9376
Session Affinity:  None
Events:            <none>
```

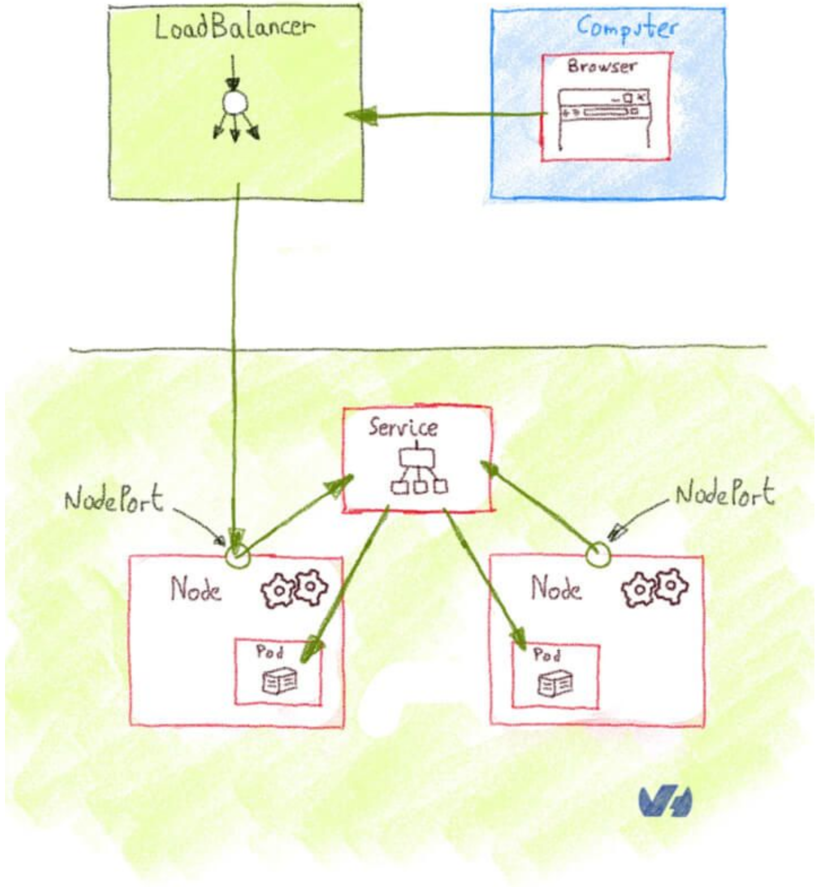
ClusterIP: Service is accessed via ‘ClusterIP’



NodePort: Service is accessed via ‘NodeIP:port’

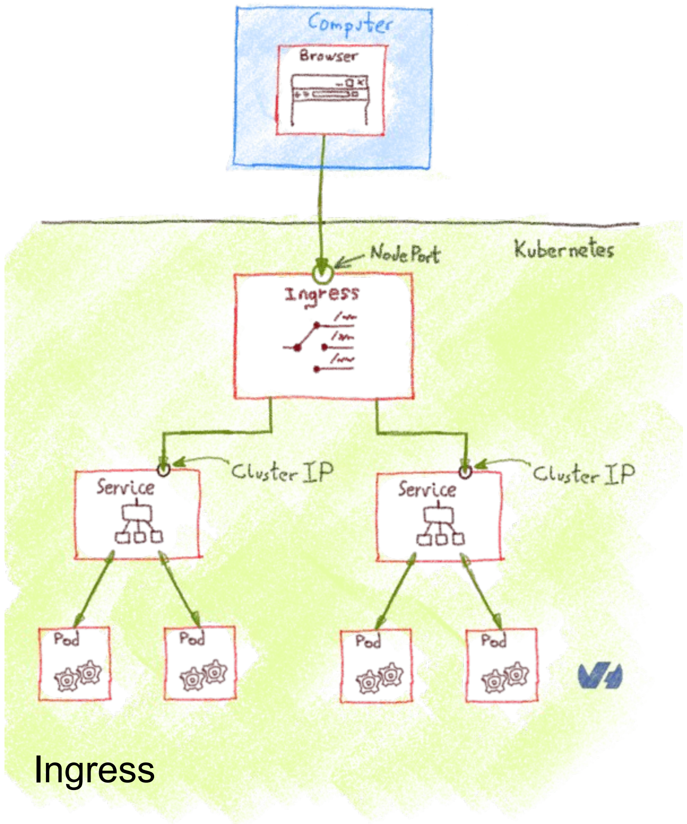


LoadBalancer: Service is accessed via Loadbalancer

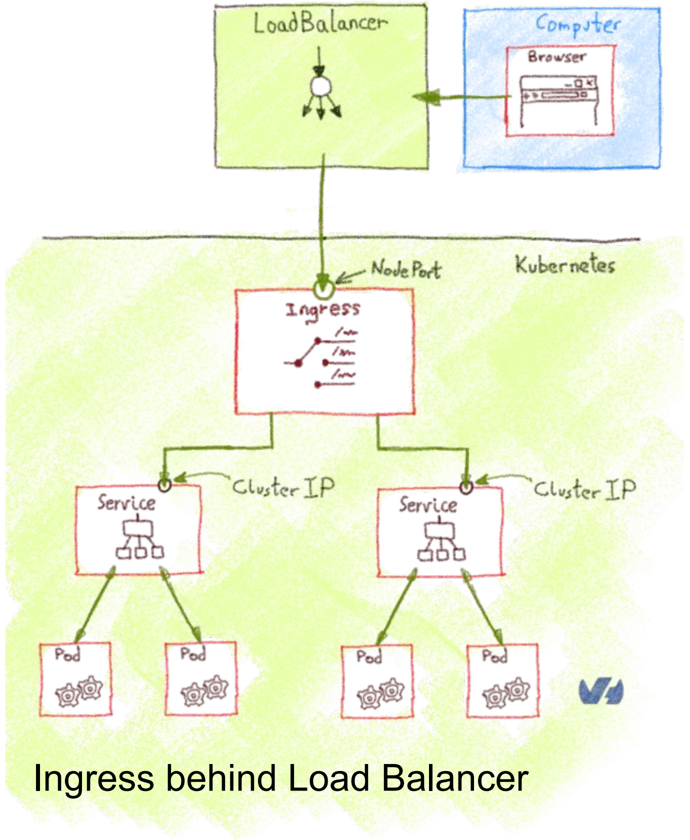


Ingress

Ingress is an API object that manages external access to the services in a cluster.



Ingress



Ingress behind Load Balancer

Part 2:

Setting up K8S The Hard Way

On top of Amazon Web Services (AWS)

1- Provisioning Compute Resources

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/03-compute-resources.md

➔ **Networking:**
- VPC
- Subnet
- Internet Gateway
- Route Tables
- Security Groups (aka Firewall Rules)

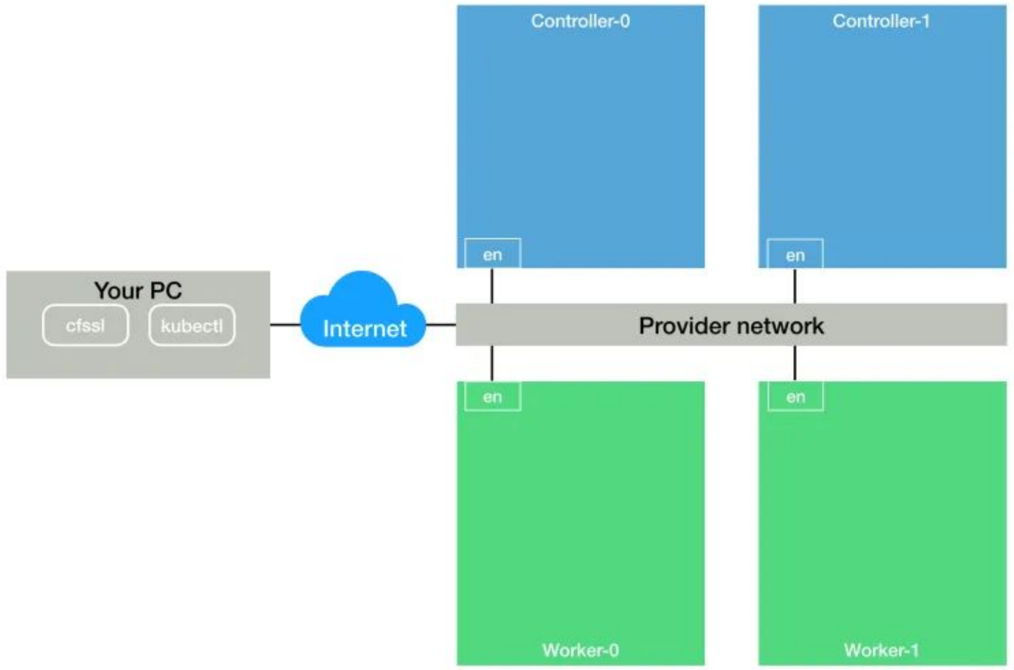
➔ Create a Network Load Balancer

➔ **Compute Instances:**

- Instance Image + SSH Key Pair
- Kubernetes Controllers
- Kubernetes Workers

1- Provisioning Compute Resources

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/03-compute-resources.md



2- Provisioning a CA and Generating TLS Certificates

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/04-certificate-authority.md

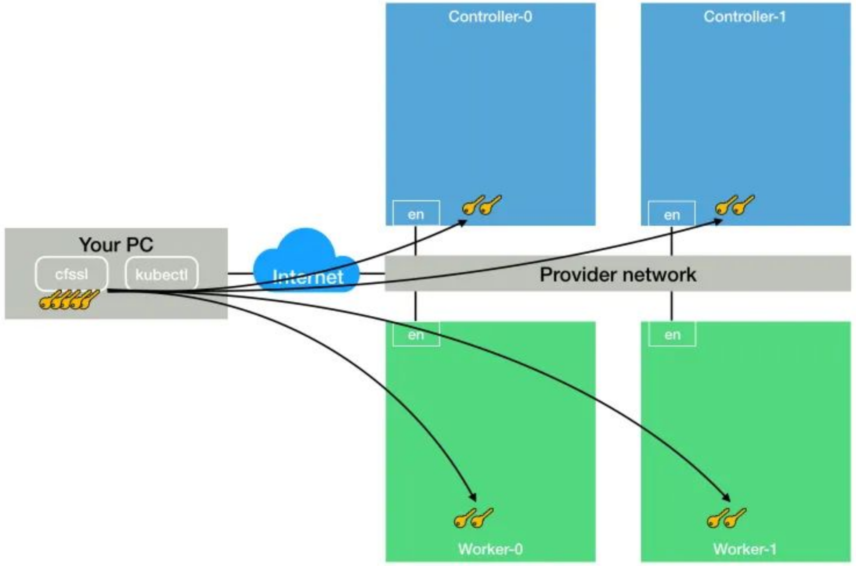
- **Certificate Authority**
- **Client and Server Certificates:**

➔ The Admin Client Certificate
➔ The Kubelet Client Certificates
➔ The Controller Manager Client Certificate
➔ The Kube Proxy Client Certificate
➔ The Scheduler Client Certificate
➔ The Kubernetes API Server Certificate

- **The Service Account Key Pair**
- **Distribute the Client and Server Certificates**

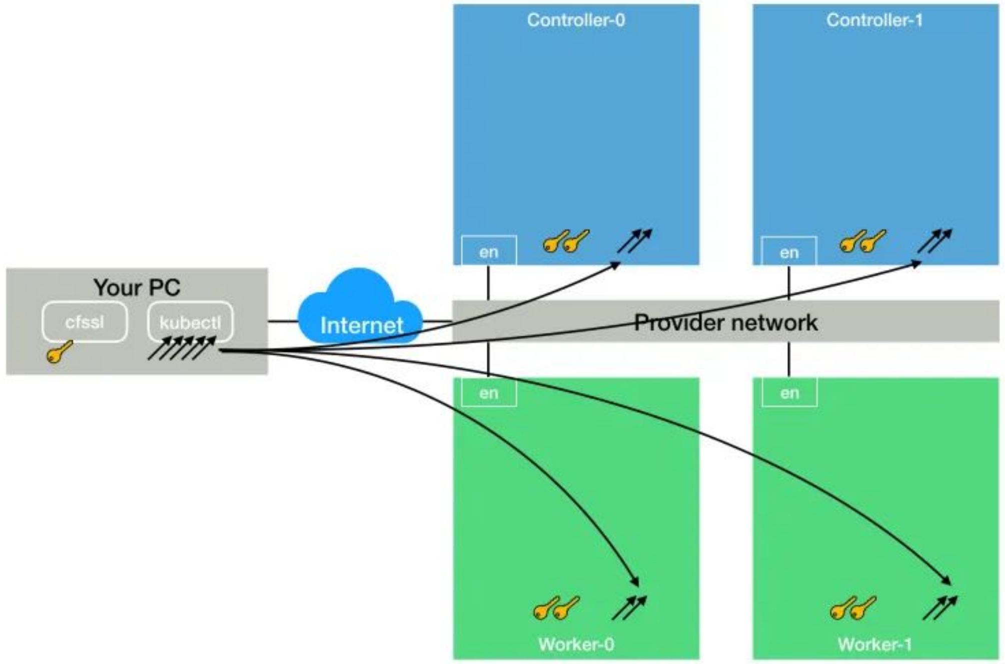
2- Provisioning a CA and Generating TLS Certificates

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/04-certificate-authority.md



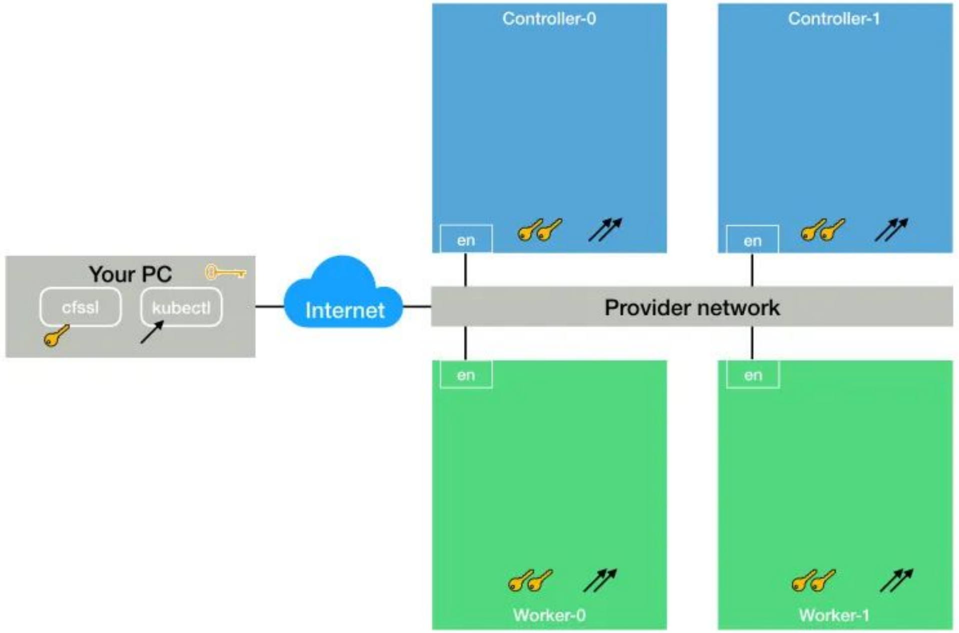
3- Generating Kubernetes Configuration Files for Authentication

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/05-kubernetes-configuration-files.md



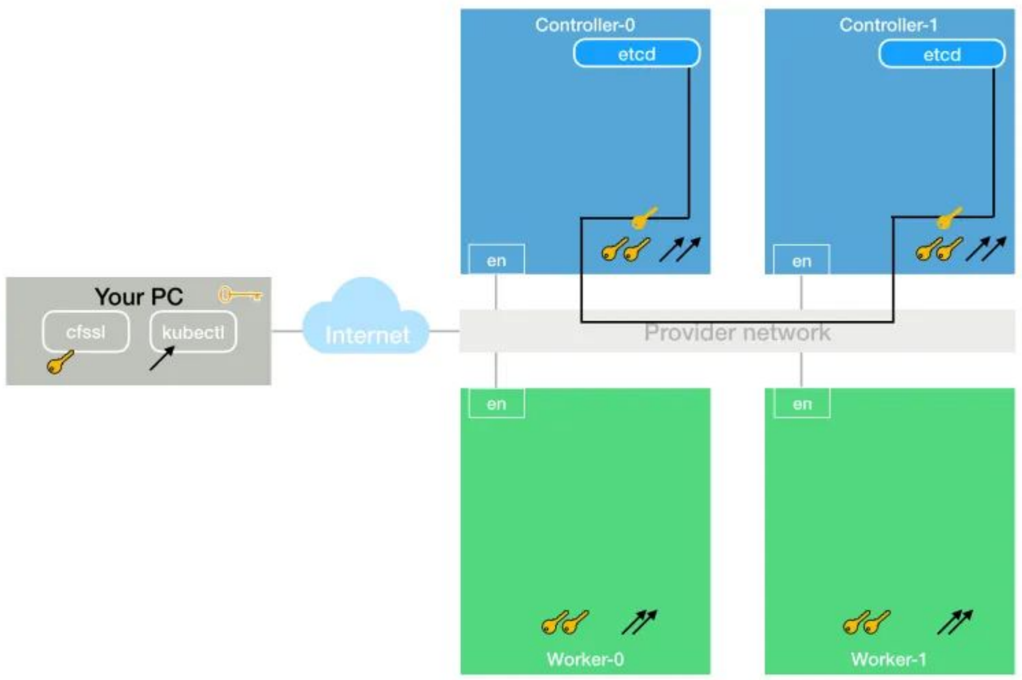
4- Generating the Data Encryption Config and Key

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/06-data-encryption-keys.md



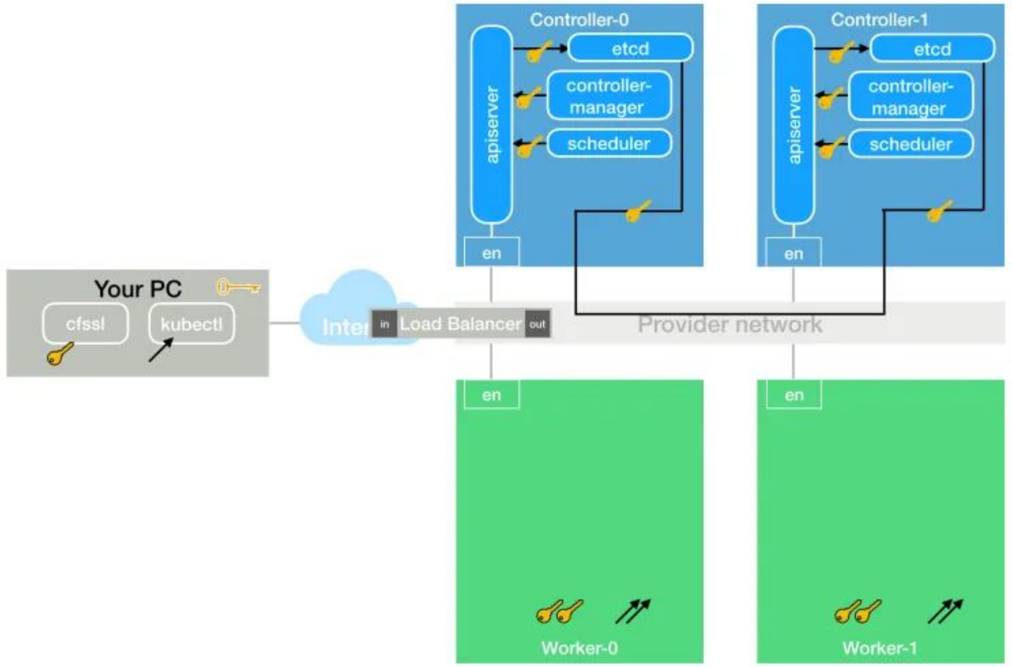
5- Bootstrapping the etcd Cluster

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/07-bootstrapping-etcd.md



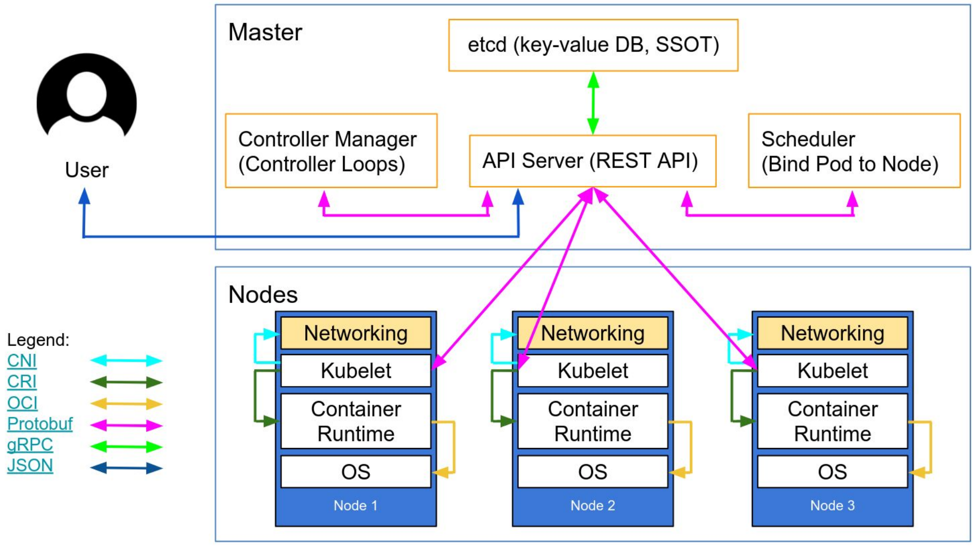
6- Bootstrapping the Kubernetes Control Plane

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/08-bootstrapping-kubernetes-controllers.md



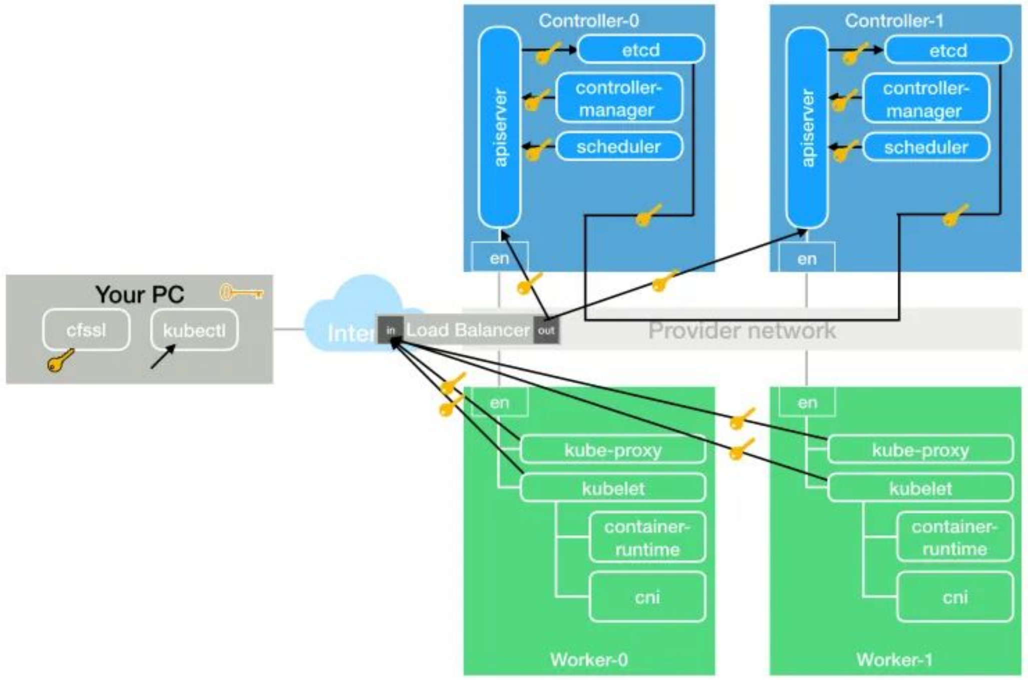
6- Bootstrapping the Kubernetes Control Plane

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/08-bootstrapping-kubernetes-controllers.md



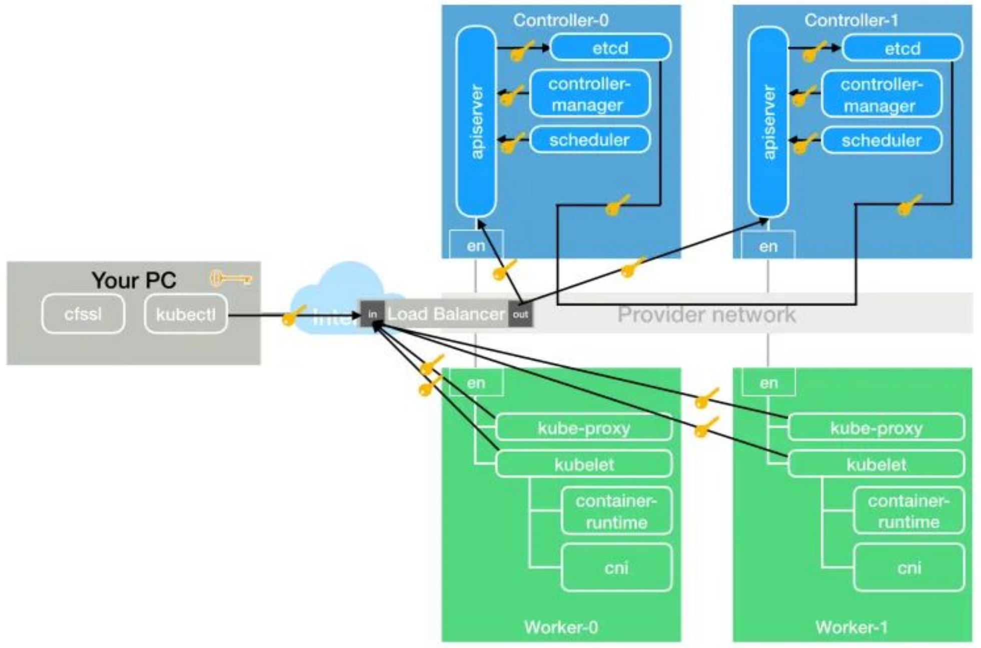
7- Bootstrapping the Kubernetes Worker Nodes

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/09-bootstrapping-kubernetes-workers.md



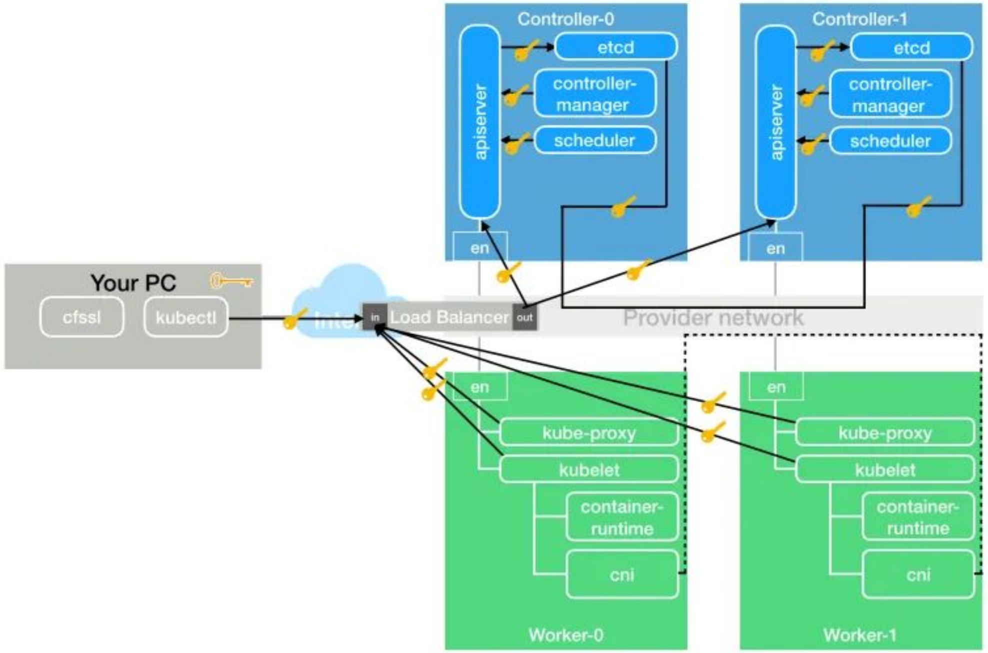
8- Configuring kubectl for Remote Access

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/10-configuring-kubectl.md



9- Provisioning Pod Network Routes

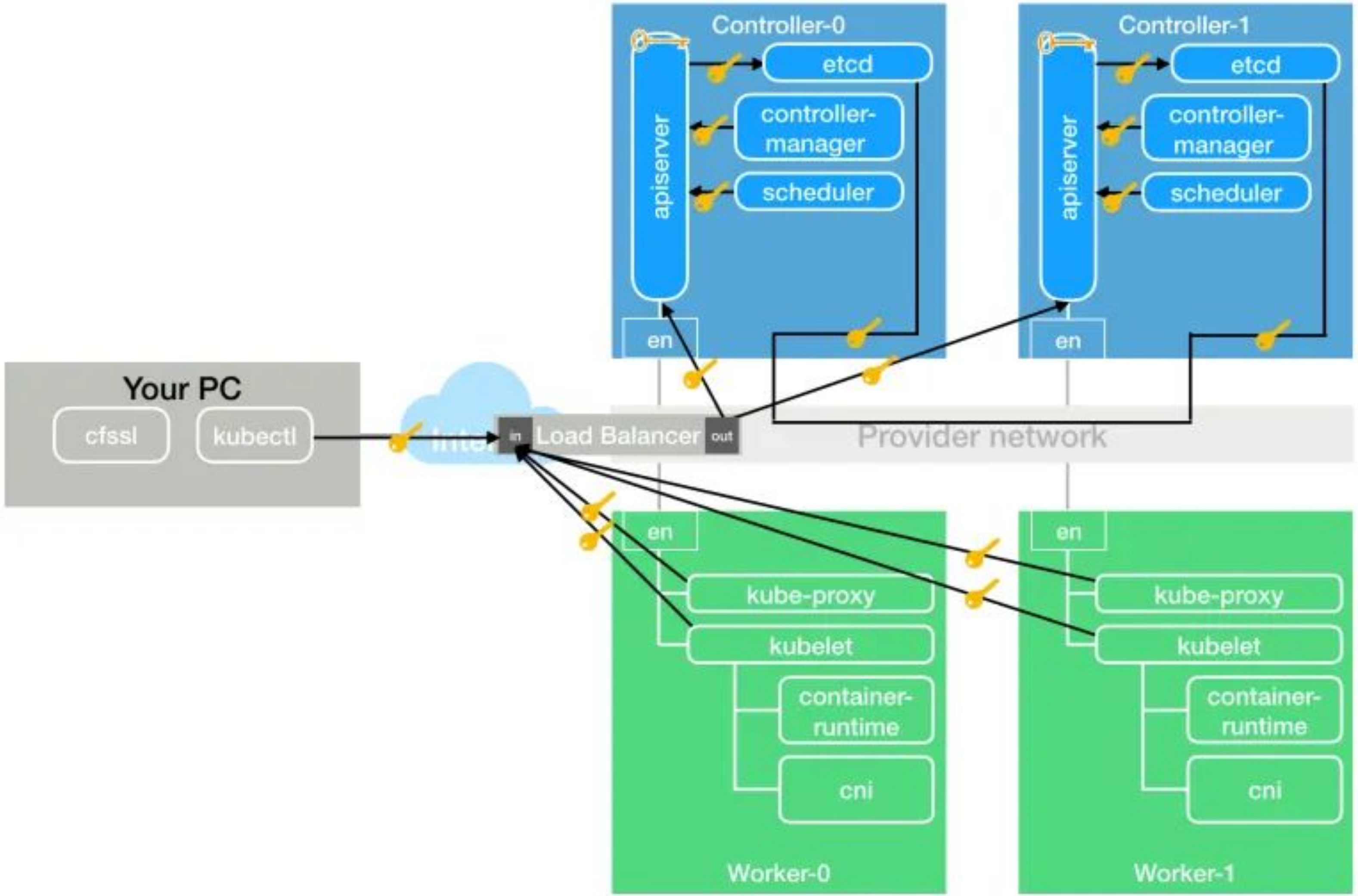
https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/11-pod-network-routes.md



10- Deploying the DNS Cluster Add-on

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/12-dns-addon.md

1


Controller-0
etcd
controller-manager
scheduler
apiserver
en
Controller-1
etcd
controller-manager
scheduler
apiserver
en
Your PC
cfssl
kubectl
Inter
in
Load Balancer
out
Provider network
en
kube-proxy
kubelet
container-runtime
cni
Worker-0
en
kube-proxy
kubelet
container-runtime
cni
Worker-1

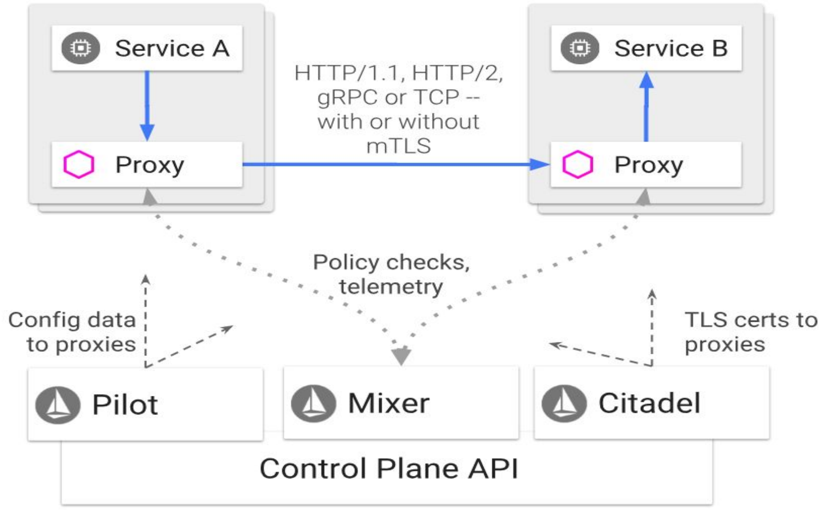
12- Cleaning Up

https://github.com/TunisJAM/kubernetes-the-hard-way-aws/blob/master/docs/14-cleanup.md

Part 4:

Istio, a modern service mesh

ISTIO architecture


Service A
Service B
HTTP/1.1, HTTP/2, gRPC or TCP -- with or without mTLS
Proxy
Proxy
Policy checks, telemetry
Config data to proxies
TLS certs to proxies
Pilot
Mixer
Citadel
Control Plane API