K8s Networking

INTRODUCTION GUIDE

September 13, 2022

Objectives and Agenda

K8s networking 101

OBJECTIVES

1. K8S COMPONENTS AND ARCHITECTURE OVERVIEW
2. K8S NETWORKING COMPONENTS OVERVIEW

AGENDA

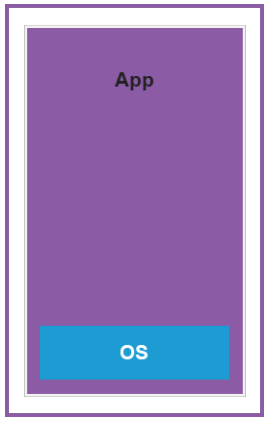
1. MODERN APPS JOURNEY
2. K8S ARCHITECTURE AND COMPONENTS
3. K8S CONCEPT
   - kubectl & kubeconfig (Lab 1.1)
   - YAML File and Pods (Lab 1.2)
   - Deployments (Lab 1.3)
   - Namespaces (Lab 1.4)
   - Services (Lab 1.5)
   - CNI (Lab 1.6)
   - DNS (Lab 1.7)
   - ClusterIP (Lab 1.8)
   - NodePort (Lab 1.9)
4. SUMMARY
5. SECTION QUIZ

MODERN APPLICATION JOURNEY

Modern Apps/DevSecOps journey

THE EVOLVEMENT OF APPLICATIONS

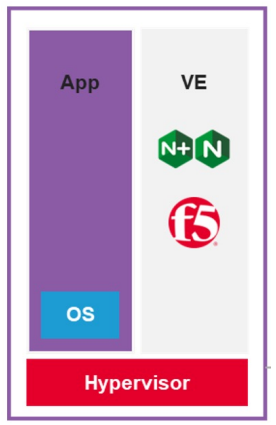
Mainframe
Before 1999



Bare Metal

- Inflexible Deployment (Overhead + Procurement process)
- Hard to Scale Up and Out

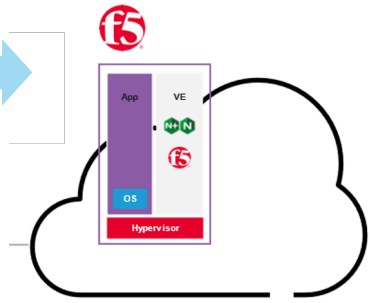
Virtualization
1999 and early 2000s



Hypervisor (VE)

- Inflexible Deployment (Procurement process)
- Hard to Scale Out

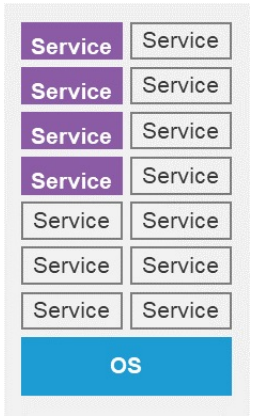
Cloud Computing
2006



Cloud Adoption

- Cloud provider dependant

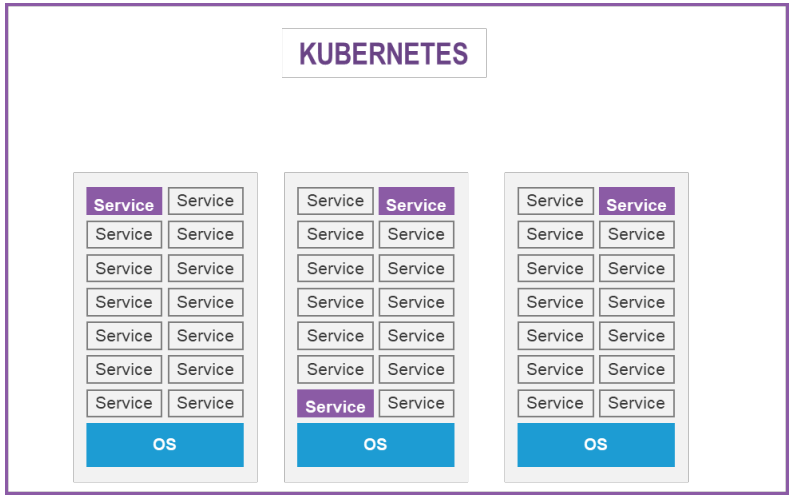
Container
2013



Docker Container

- Flexible Deployment with less overhead
- Cloud Agnostics
- No Management Control (single host)

Kubernetes (Google)
2014



Kubernetes

- Flexible Deployment with less overhead
- Cloud Agnostics
- Kubernetes as the management plane to control all the deployments (Multiple Hosts)

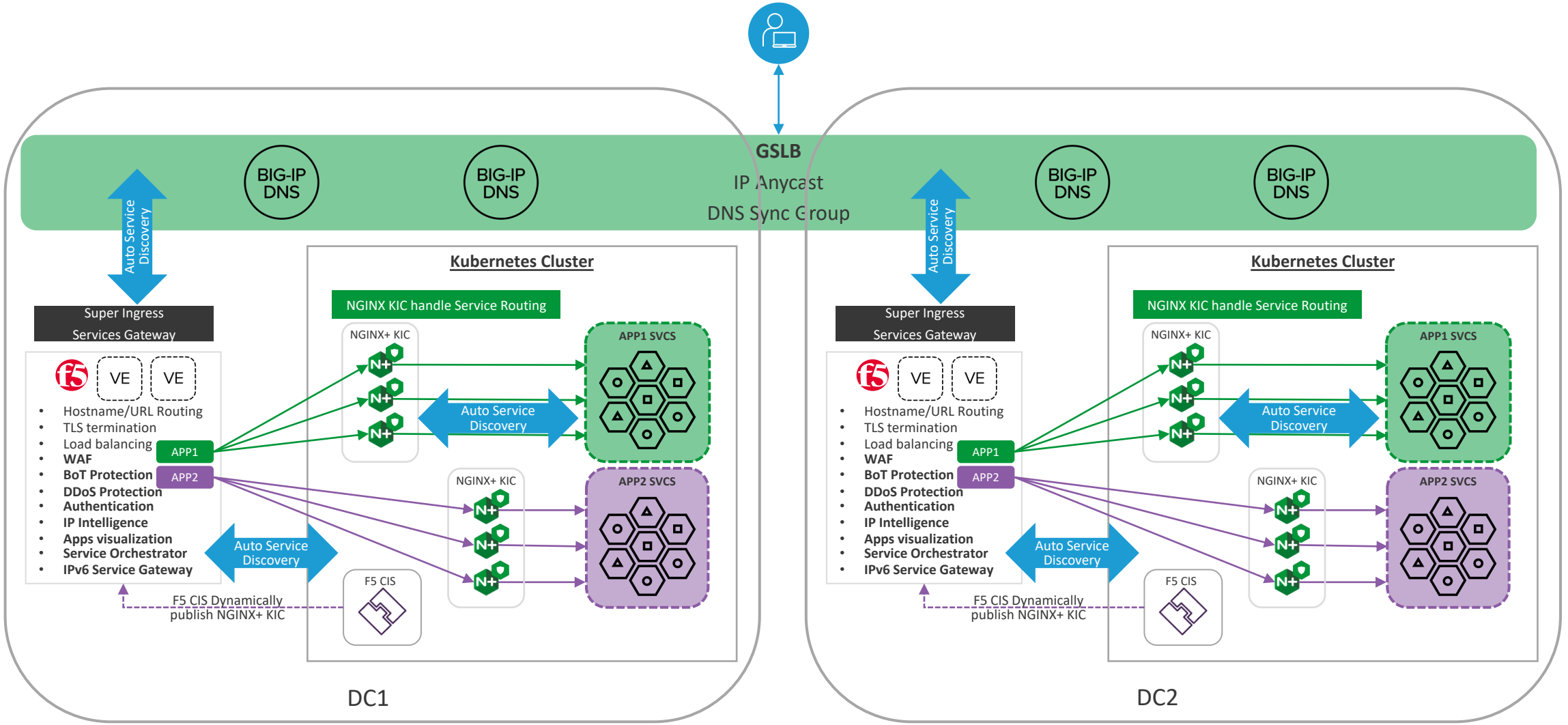
Traditional Monolith App — Cloud Native / Micro-services

Application Landscape

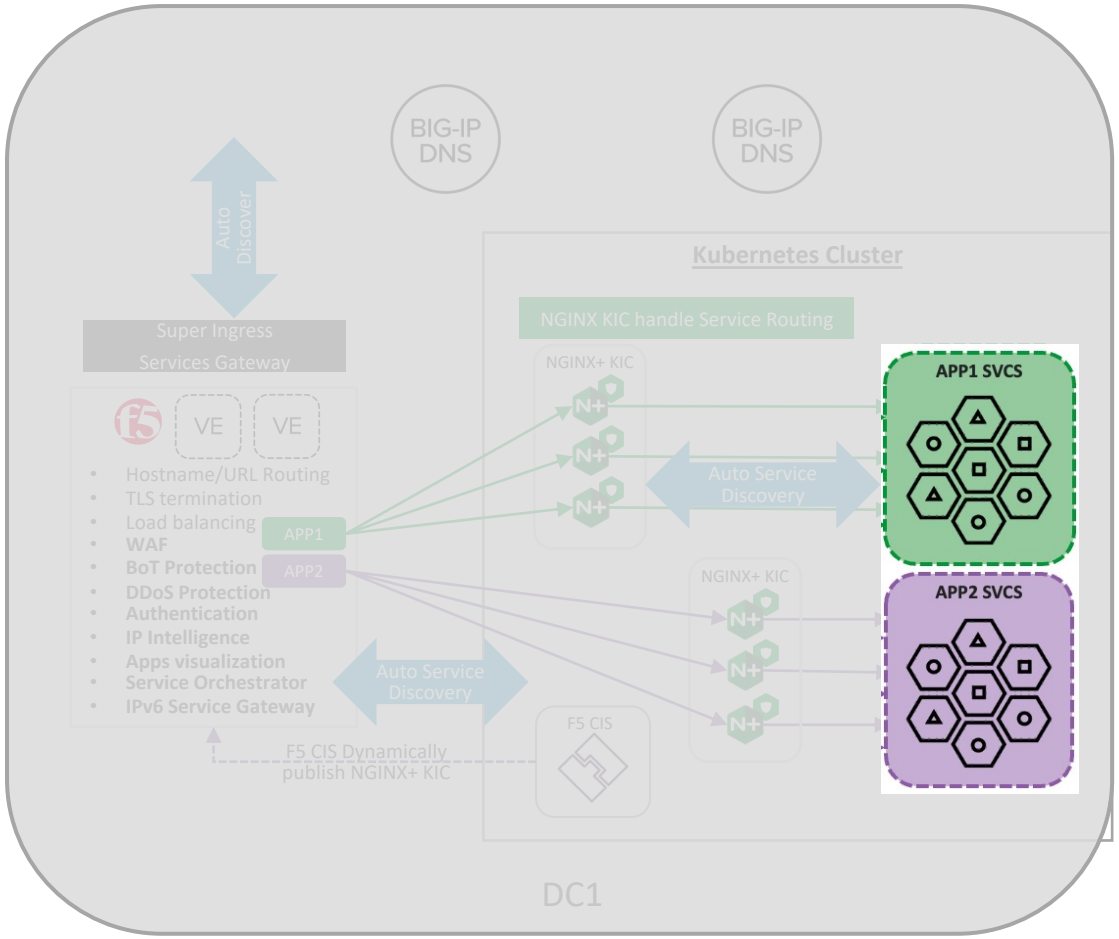
Centralized — De-centralized / Edge Deployment

Application Deployment

Agenda : End-goal with Active-Active K8s Clusters Architecture



Day 1 Agenda



Chapter 1 : Modern Applications and DevOps Basics (Theory + Lab)

- Legacy vs VM vs Container (Microservices example in Real Life)
- Getting comfortable with Command Line Editors
  - SSH Commands
  - Using VI Editor (Delete entire line, save, undo, etc)
  - Visual Studio (e.g., F5/NGINX Plug-in)

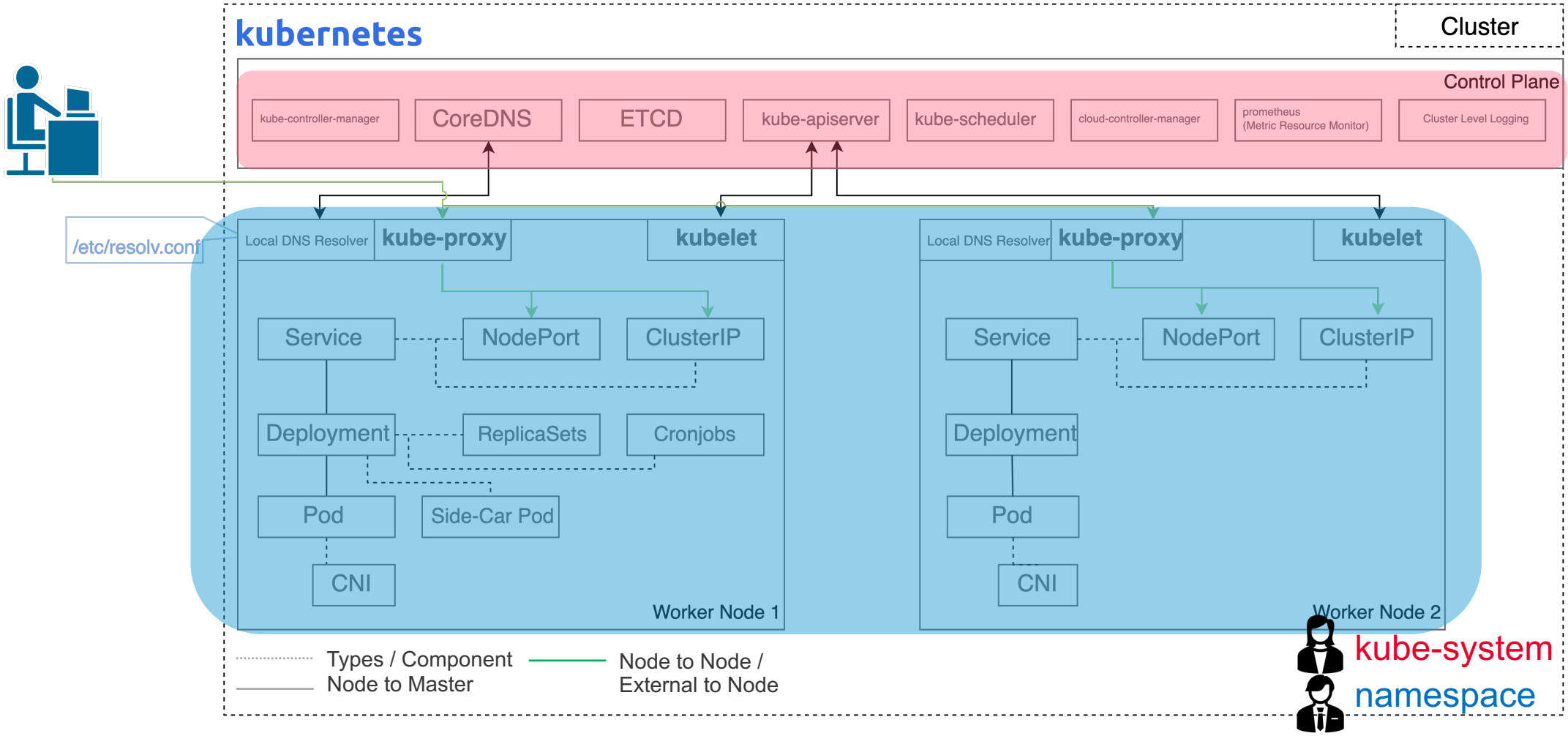
Chapter2 : Kubernetes for Beginners (Theory + K8s Lab)

- Containers and Pods
- YAML Basics – How to read K8s manifests
- K8s Deployment
- K8s Networking: ClusterIP, Services, LB, NodePort

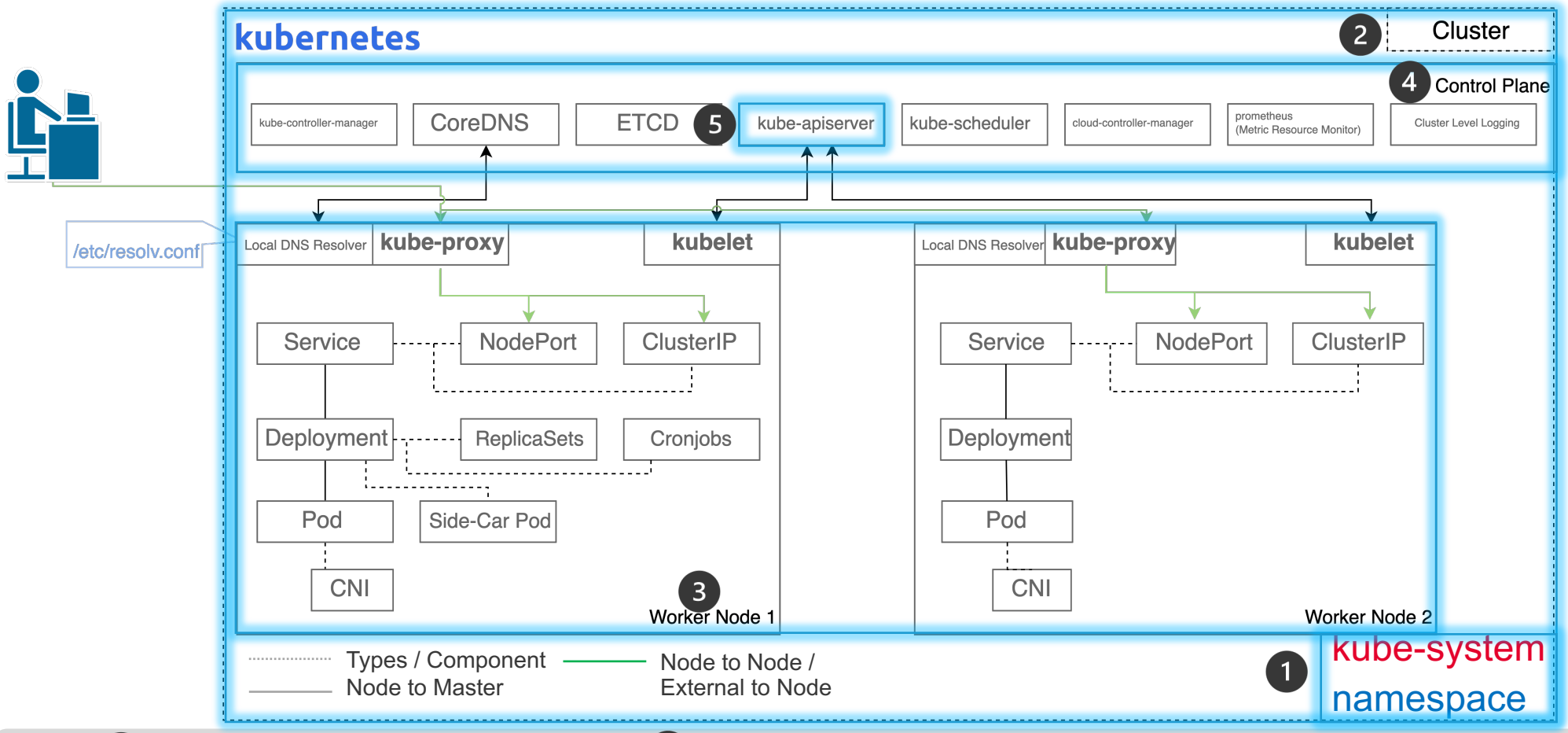
Chapter3: K8s and NGINX Ingress Controller Integration (Theory + Lab)

- What is Ingress Controller?
- How does NGINX+ look like as Ingress Controller?
- NGINX+ value, strength and positioning

Typical K8s deployment for microservices



K8s Topology: Cluster Overview & Master Node



1 **Namespace**: A way to divide cluster resources between multiple users

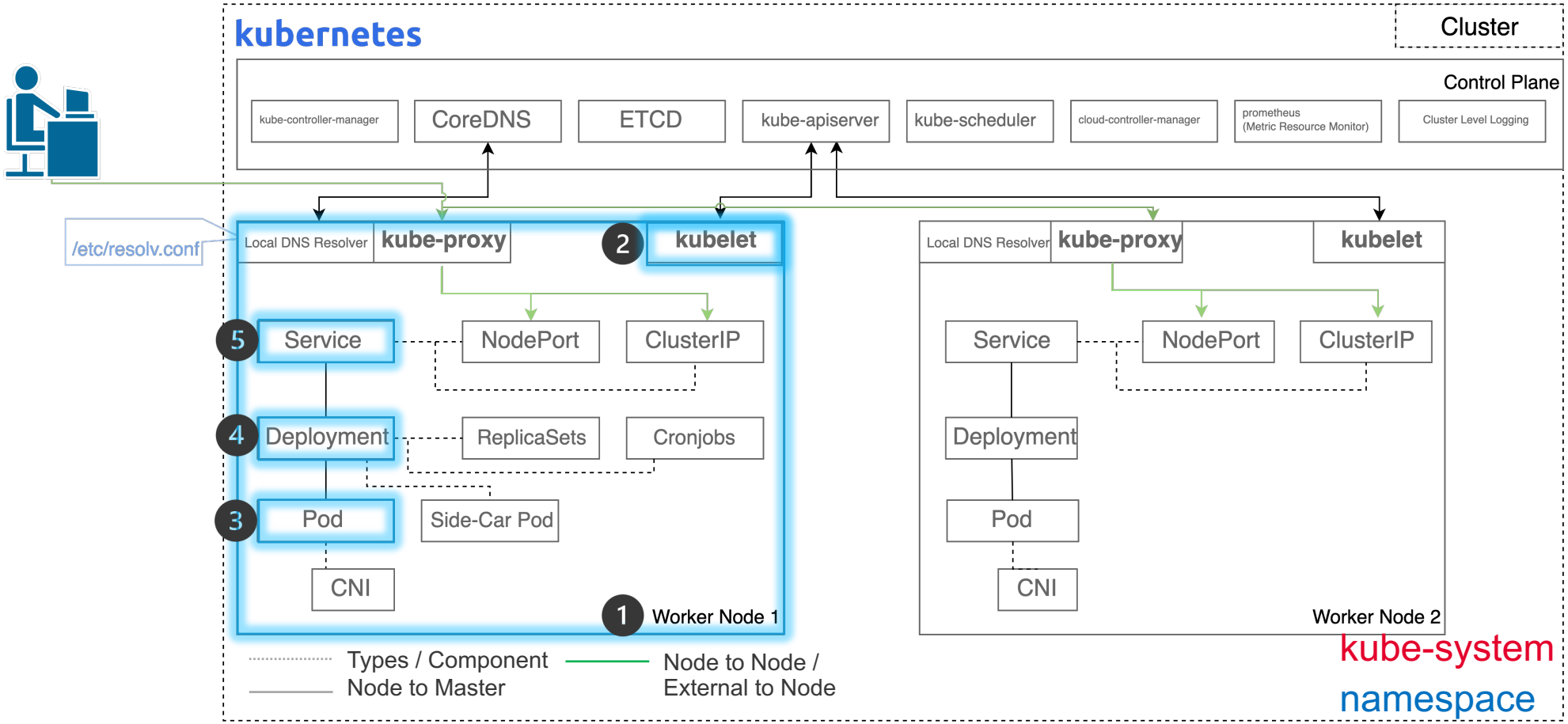
2 **Cluster**: Pool of Nodes working together intelligently

3 **Node**: Representation of a virtual machine

4 **Control Plane (Master Node)**: responsible for cluster management like workload distribution and state of the cluster

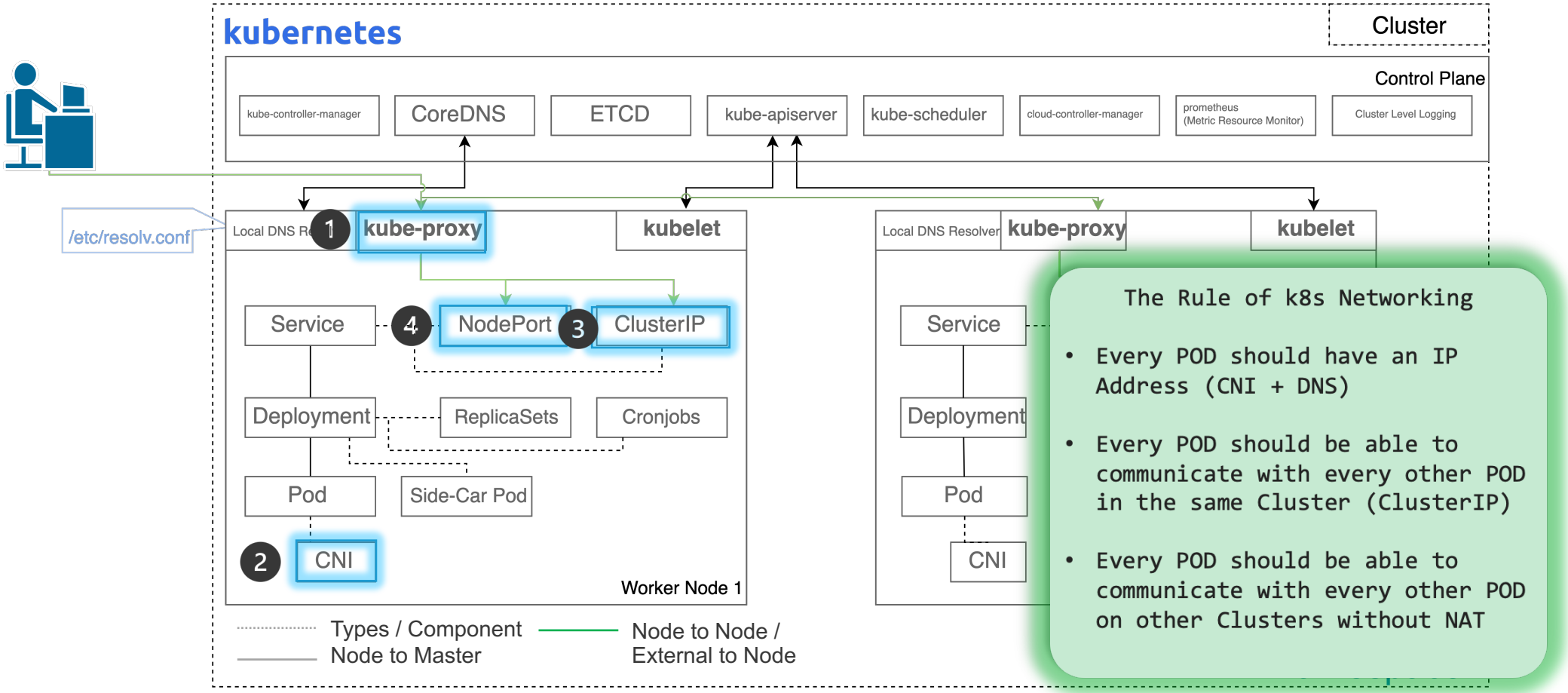
5 **kube-apiserver**: API Gateway for operators to the cluster. Used to manipulate each node's components via the usage of API.

K8s Topology: Worker Node



1. **Worker Node**: runs the workloads for the master node.
2. **Kubelet**: is an agent that helps the master node to ensure the pods are running within a node
3. **Pod**: runs the application itself in the form of containers
4. **Deployment**: used as instruction to inform the Kubelet how to create or modify pods in a controlled manner
5. **Service**: used to expose a set of deployments to be consumed by other pods within or outside the cluster. (More on that in the next section)

K8s Topology: Network Connectivity



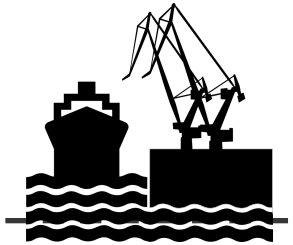
The Rule of k8s Networking

- Every POD should have an IP Address (CNI + DNS)
- Every POD should be able to communicate with every other POD in the same Cluster (ClusterIP)
- Every POD should be able to communicate with every other POD on other Clusters without NAT

1. **kube-proxy**: a network proxy that runs on each node in your cluster. It **maintains network rules on nodes.**
2. **Container Network Interface:** Allows the node to be identified in the network
3. **ClusterIP**: Communication of pods in the same cluster
4. **NodePort**: Exposing Cluster to external

K8s Illustrated

K8S ILLUSTRATED IN A SHIP SUPPLY CHAIN



Port

Records

Lookup

etcd

c-m

api

Port Operations Manager

Crane

Instruct

sched.

Load - Unloading

Captain

Cargo Ship

master

Port Control Tower

Maintains communication

kubelet

k-proxy

Captain

Cargo Ship

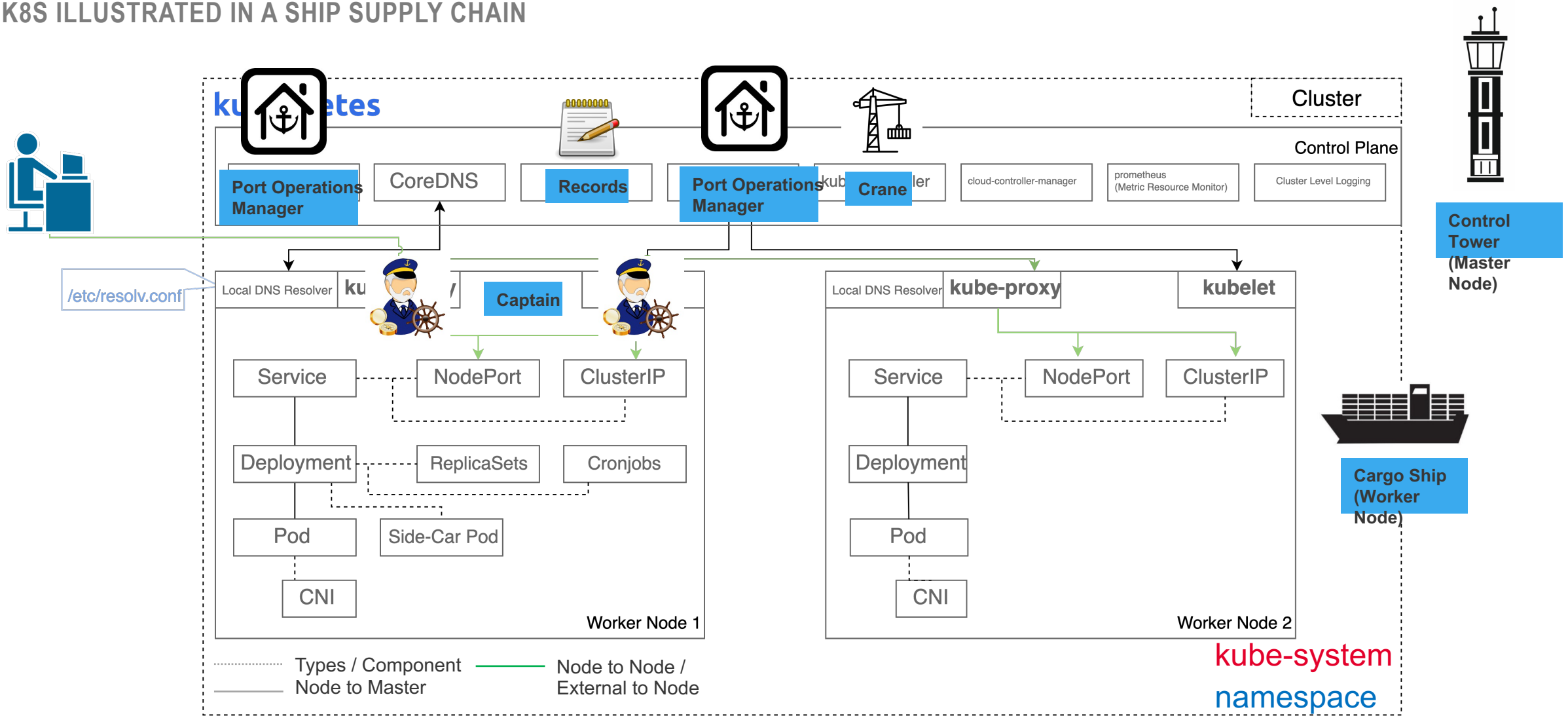
node

Captain

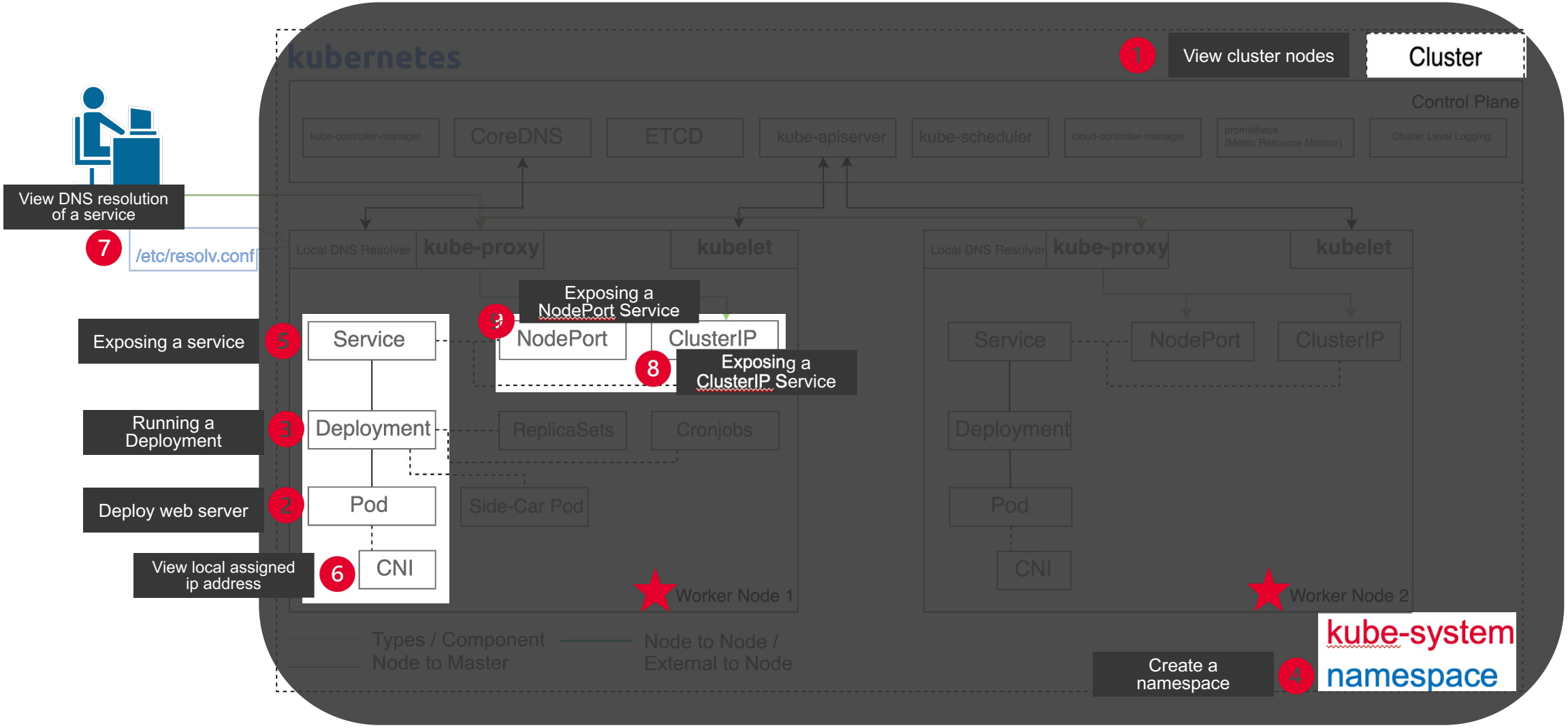
Cargo Ship

K8s Illustrated

K8S ILLUSTRATED IN A SHIP SUPPLY CHAIN



Lab Task Overview



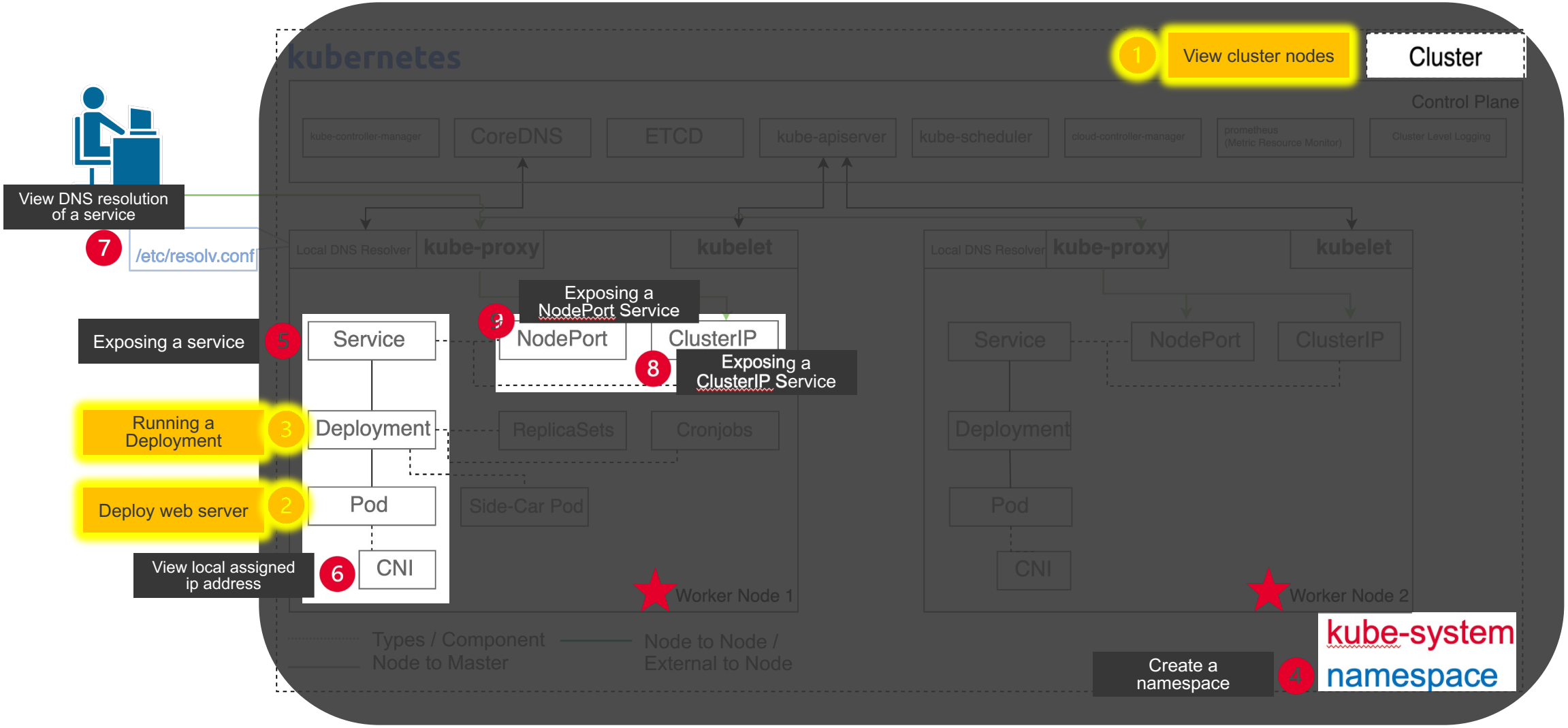
Lab Guide Introduction and spinning of labs

10 Mins

Kubernetes Concepts

kubectl and kubeconfig
yaml files and pods
deployments
namespaces
services

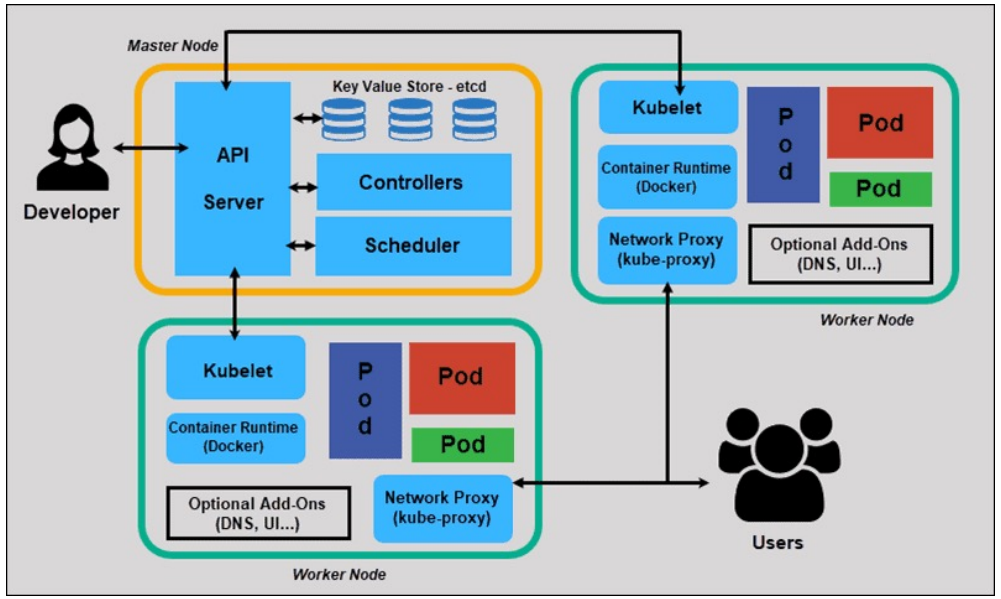
Lab Task Overview



K8s Concept – Lab 1.1

KUBECTL AND KUBECONFIG

- **Kubectl** is the command for developer to make a call to API Server (Master Node)
- In the environment, where K8s master nodes are more than one, the API endpoint can be fronted by a load-balancer to load-balance between the master nodes
- Commands to test the API call and Environment,
  - **kubectl get nodes** ~> List of all nodes
  - Use option **-o wide** to give more information
  - **kubectl get nodes -o wide**




```
    server: https://10.1.1.10:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
```

K8s Concept – Lab 1.2

YAML (YAH-ML) FILES AND PODS

- **YAML** is a human-readable data-serialization language used for configuration purpose
- Although **kubectl** command can do many things, in some complex scenarios using YAML file is easier for configuration management
- Commands to create a pods using kubectl,
  - **kubectl run nginx-kubectl --image=nginx --restart=Never**
  - **kubectl get pods –o wide** ~> to see pods information
  - **kubectl delete pods nginx-kubectl**
- Commands to create a pods using yaml,
  - **kubectl apply –f <filename>.yaml**
  - **kubectl get pods –o wide** ~> to see pods information
  - **kubectl delete pods nginx**

Example of YAML File

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
```

K8s Concept – Lab 1.3

DEPLOYMENTS

- **Deployment** defines a pod's desired behavior or characteristics.

- In layman terms, differences between Deployment and Pods are,
  - Define Replica
  - Upgrade Pods and versioning
  - Rollback and so many others

- In production, people will use Deployment rather than pods

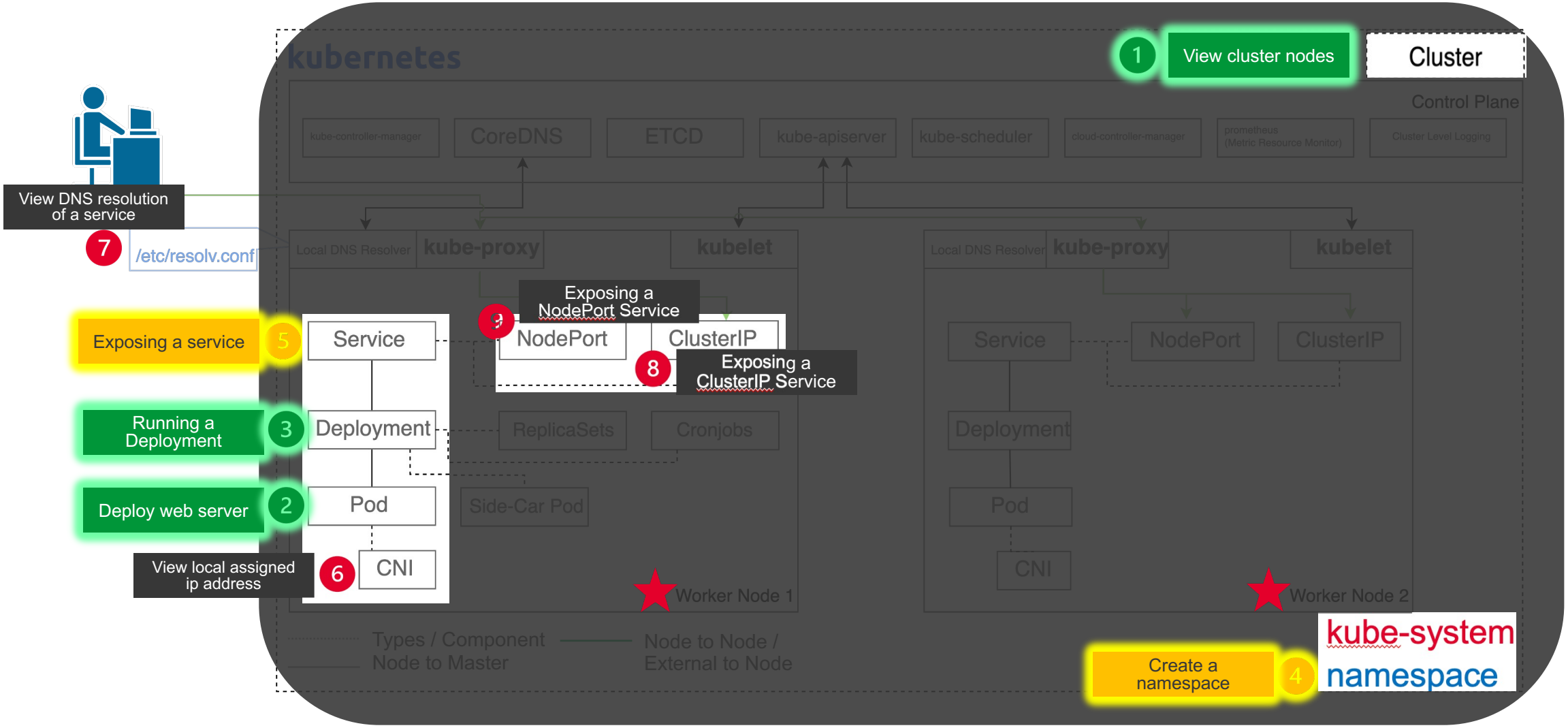
- Commands to create a deployment,
  - **kubectl apply –f <filename>.yaml**
  - **kubectl get deployment –o wide** ~> to see deployment's information
  - **kubectl describe deployment nginx-deployment**
  - **kubectl delete deployment nginx-deployment**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Lab Time (Lab 1.1 – 1.3)

20 Mins

Lab Task Overview



K8s Concept – Lab 1.4

NAMESPACES

- **Namespace** provides a mechanism for isolating groups of resources within a single cluster.

- There are a couple of default namespaces, https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/
  - Default
  - kube-system
  - kube-public
  - kube-node-lease

- Commands to create a pod in the namespace,
  - **kubectl apply –f <filename>.yaml**
  - **kubectl get pods**
  - **kubectl get pods –n production**
  - **kubectl describe pods mypod –n production**
  - **kubectl delete pods mypod -n production**

```
apiVersion: v1
kind: Namespace
metadata:
  name: production
  labels:
    name: production
---
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  namespace: production
  labels:
    name: mypod
spec:
  containers:
  - name: mypod
    image: nginx
```

K8s Concept – Lab 1.5

SERVICES

- **Services** is an abstract way to expose an application running on a set of Pods as a network service
- By default, cluster IP
- Commands to create a pod and expose the service,
  - **kubectl apply –f <filename>.yaml**
  - **kubectl get pods**
  - **kubectl get service**
  - **kubectl describe pods**
  - **kubectl describe service**
  - **Try to access the pods using curl from jumphost**

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  type: NodePort
  ports:
    - port: 80
      nodePort: 30080
      name: http
selector:
    name: nginx
```

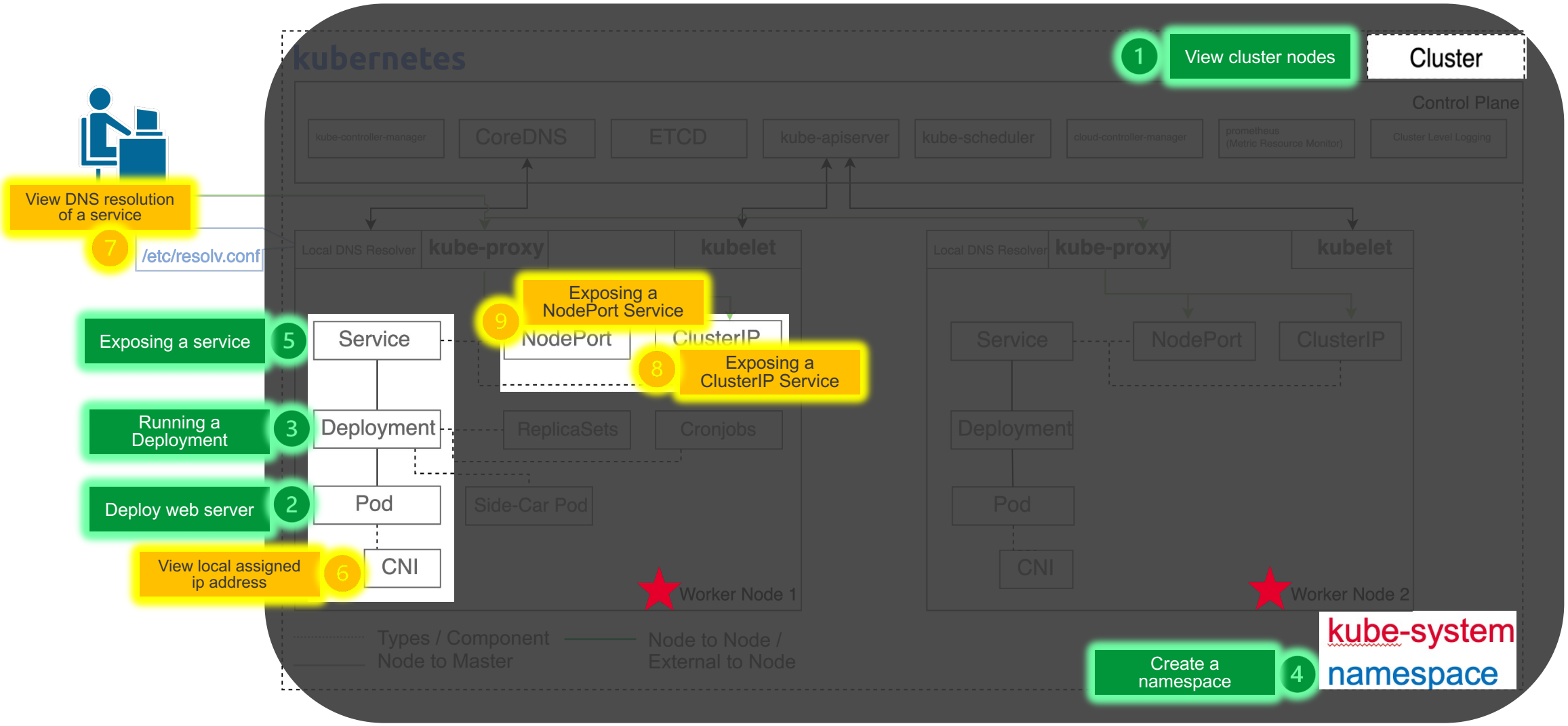
Lab Time (Lab 1.4 – 1.5)

20 Mins

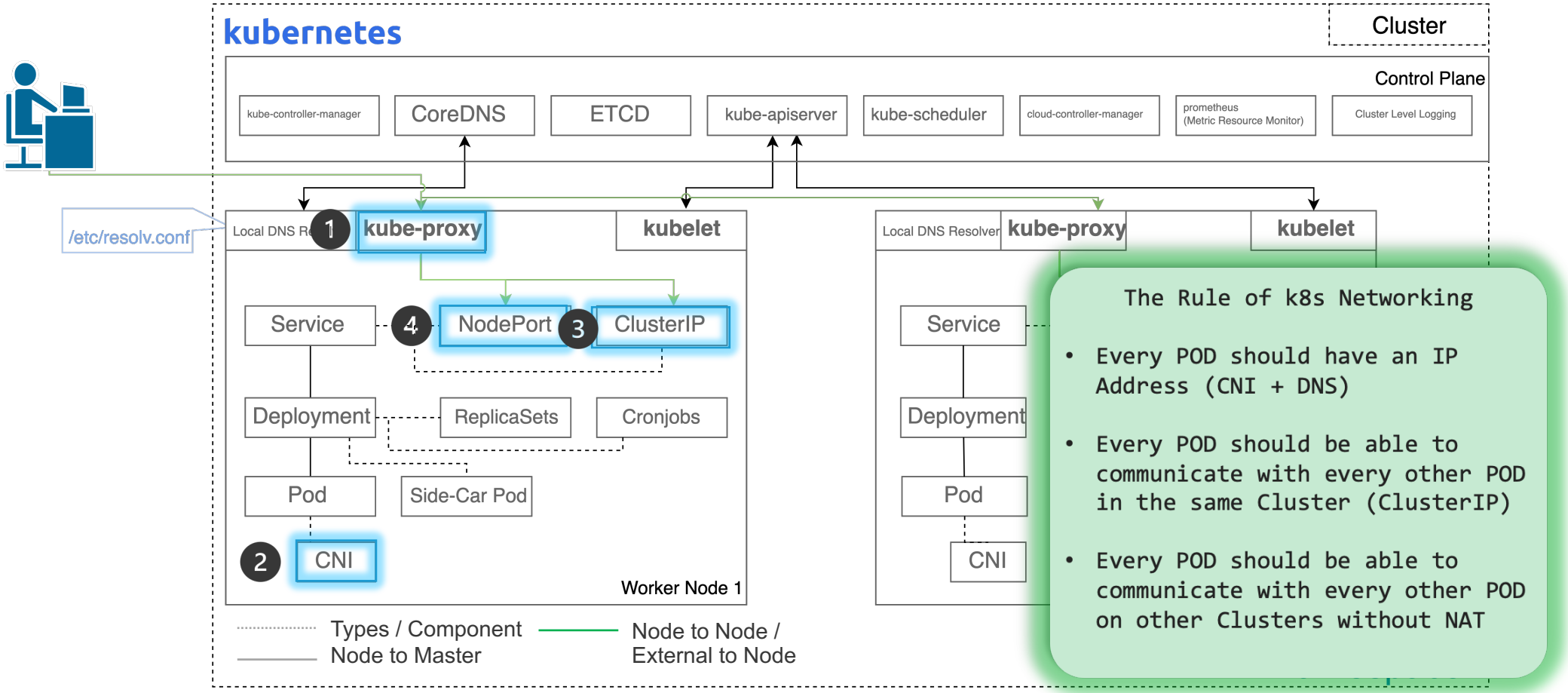
Kubernetes Concepts

Container Network Interfaces
DNS
ClusterIP
NodePort

Lab Task Overview



K8s Topology: Network Connectivity



The Rule of k8s Networking

- Every POD should have an IP Address (CNI + DNS)
- Every POD should be able to communicate with every other POD in the same Cluster (ClusterIP)
- Every POD should be able to communicate with every other POD on other Clusters without NAT

1. **kube-proxy**: a network proxy that runs on each node in your cluster. It **maintains network rules on nodes.**
2. **Container Network Interface:** Allows the node to be identified in the network
3. **ClusterIP**: Communication of pods in the same cluster
4. **NodePort**: Exposing Cluster to external

K8s Concept – CNI

The Rule of K8s Networking

- 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

K8 Networking

- **Container Network Interface** is a plugin that helps to configure the network, provision IP addresses, and maintain connectivity with multiple hosts
- The configuration to setup the IP subnet being used is defined in /etc/Kubernetes/manifests/kube-controller-manager.yaml
- Commands to be executed to know the IP address of a pods
  - **kubectl describe pods <pods' name>**
  - **kubectl get pods –o wide**

```
ubuntu@jumphost:~$ kubectl get pods -o wide
NAME                              READY   STATUS    RESTARTS   AGE     IP            NODE
nginx-yaml                        1/1     Running   1          7d13h   10.244.3.65   worker1
source-ip-app-86cd545f4c-q95nf    1/1     Running   2          7d23h   10.244.4.75   worker2
```

```
ubuntu@jumphost:~$ kubectl describe pods nginx-yaml
Name:         nginx-yaml
Namespace:    default
Priority:     0
Node:         worker1/10.1.1.7
Start Time:   Tue, 05 Jul 2022 02:17:24 +0000
Labels:       name=nginx-yaml
Annotations:  <none>
Status:       Running
IP:           10.244.3.65
IPs:
  IP:  10.244.3.65
```

```
root@master1:/etc/kubernetes/manifests# cat kube-controller-manager.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-controller-manager
    tier: control-plane
  name: kube-controller-manager
  namespace: kube-system
spec:
  containers:
  - command:
    - --experimental-cluster-signing-duration=87600h
    - kube-controller-manager
    - --allocate-node-cidrs=true
    - --authentication-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --authorization-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --bind-address=127.0.0.1
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --cluster-cidr=10.244.0.0/16
    - --cluster-name=kubernetes
    - --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
    - --port=0
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --root-ca-file=/etc/kubernetes/pki/ca.crt
    - --service-account-private-key-file=/etc/kubernetes/pki/sa.key
    - --service-cluster-ip-range=10.96.0.0/12
    - --use-service-account-credentials=true
    image: k8s.gcr.io/kube-controller-manager:v1.21.1
```

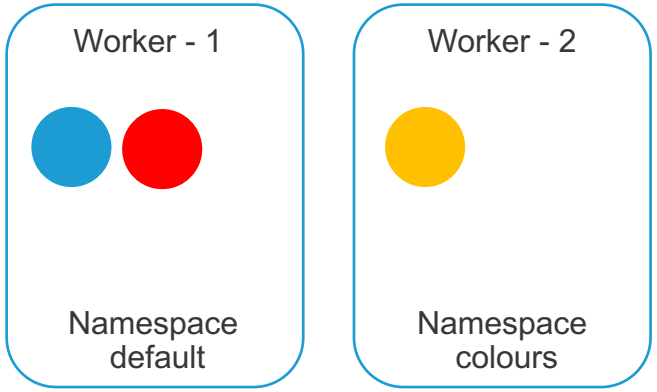
K8s Concept

CORE DNS CONCEPT

- **Core DNS** providing a way to call the pods or services inside K8s via DNS name

The Rule of K8s Networking

- 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



Services Name	IP address
Blue	10.244.1.5
Red	10.244.5.5
Yellow	10.107.37.1

With DNS Calling Services

```
curl http://red/
```

```
red.default.svc.cluster.local
```

```
curl http://yellow.colours/
```

```
yellow.colours.svc.cluster.local
```

Without DNS

```
curl http://10.244.5.5/
```

```
curl http://10.107.37.1/
```

With DNS Calling pods

```
Ping 10-244-5-5.default.pod.cluster.local
```

```
Ping 10-107-37-1.colours.pod.cluster.local
```

K8s Concept

CORE DNS IS A DEFAULT MODULE (PART OF KUBE-SYSTEM)

- **Recap: Namespace**
- **Namespace** is a mechanism for isolating groups of resources within a single cluster.
- All out of the box modules will be found in **kube-system**
- Use parameter –n kube-system
- Use parameter --all-namespaces

```
kuubuntu@jumphost:~$
ubuntu@jumphost:~$ kubectl get namespace
NAME              STATUS   AGE
default           Active   390d
kube-node-lease   Active   390d
kube-public       Active   390d
kube-system       Active   390d
nms               Active   77d
```

The Rule of K8s Networking

- 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

```
ubuntu@jumphost:~$ kubectl get pods -n kube-system
NAME                                   READY   STATUS             RESTARTS   AGE
coredns-558bd4d5db-57nt6               1/1     Running            32         390d
coredns-558bd4d5db-9h5cz               1/1     Running            36         390d
etcd-master1                           1/1     Running            120        390d
etcd-master2                           1/1     Running            120        390d
etcd-master3                           1/1     Running            117        390d
kube-apiserver-master1                 1/1     Running            159        390d
kube-apiserver-master2                 1/1     Running            149        390d
kube-apiserver-master3                 1/1     Running            149        390d
kube-controller-manager-master1        0/1     CrashLoopBackOff   223        16d
kube-controller-manager-master2        1/1     Running            37         390d
kube-controller-manager-master3        1/1     Running            37         390d
kube-flannel-ds-5rjn7                  1/1     Running            34         390d
kube-flannel-ds-cl9pk                  1/1     Running            38         390d
kube-flannel-ds-fkbv9                  1/1     Running            34         390d
kube-flannel-ds-grh22                  1/1     Running            33         390d
kube-flannel-ds-lrrhg                  1/1     Running            38         390d
kube-proxy-5v4mg                       1/1     Running            33         390d
kube-proxy-qpj5c                       1/1     Running            32         390d
kube-proxy-rl7fg                       1/1     Running            32         390d
kube-proxy-t84x5                       1/1     Running            36         390d
kube-proxy-v5pb7                       1/1     Running            38         390d
kube-scheduler-master1                 1/1     Running            46         390d
kube-scheduler-master2                 1/1     Running            41         390d
kube-scheduler-master3                 1/1     Running            36         390d
nfs-client-provisioner-7bf658486d-dk8pg 1/1    Running            30         238d
```

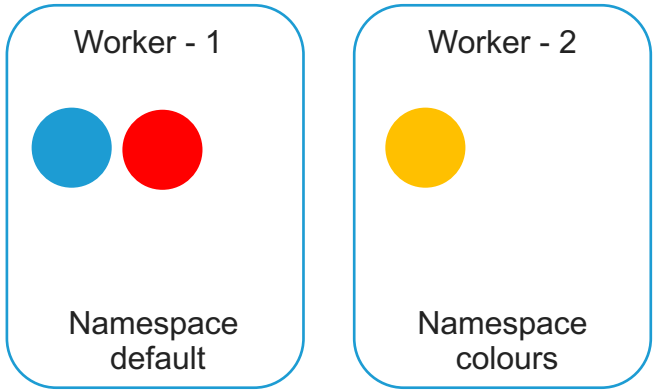
K8s Concept – DNS

CORE DNS NAMING CONVENTION

- There is a specific naming convention in K8s to call a service or pods in K8s

```
<Name Service/Pods>.<Namespace>.<Type(pod/svc)>.<Root Domain>
```

- All pods within namespace however, can call the service or pods directly without needing to call FQDN



Services Name	IP address
Blue	10.244.1.5
Red	10.244.5.5
Yellow	10.107.37.1

With DNS Calling Services

```
curl http://red/
```

```
red.default.svc.cluster.local
```

```
curl http://yellow.colours/
```

```
yellow.colours.svc.cluster.local
```

With DNS Calling pods

```
Ping 10-244-5-5.default.pod.cluster.local
```

```
Ping 10-107-37-1.colours.pod.cluster.local
```

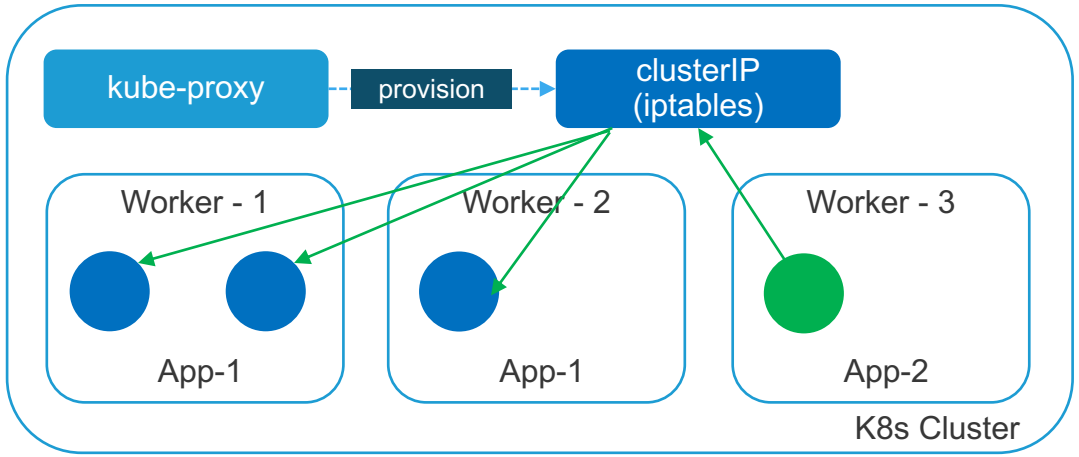
- Commands to get into bash,
  - **kubectl exec -it <pod name> -n <namespace> bash**
  - For the lab
    - Spin up pods image busybox42/alpine-pod and go into the pods
      kubectl run alpine --image=busybox42/alpine-pod --restart=Never
    - Do nslookup for services that is being provisioned in previous lab
    - Bonus: Try to ping the pods that is being defined in the previous lab using DNS name

K8s Concept - ClusterIP and NodePort

EXPOSING SERVICE (CLUSTER IP AND NODEPORT) CONCEPT

Why we need service if we can go to the pods directly?

Cluster IP
Internal pods to pods communication (EW traffic)



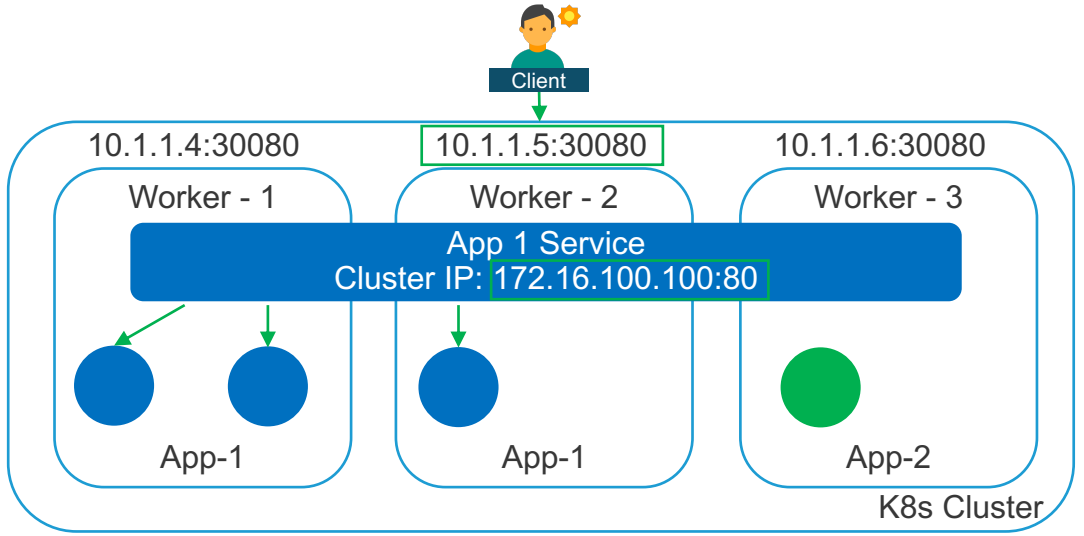
- For each Service, **kube-proxy installs iptables rules**, which capture traffic to the Service's clusterIP and port, and redirect that traffic to one of the Service's backend sets.

- **By default, kube-proxy in iptables mode chooses a backend at random.** There are different kube-proxy modes available

```
ubuntu@jumphost:~$ kubectl logs kube-proxy-5v4mg -n kube-system
I0713 08:14:12.902730       1 node.go:172] Successfully retrieved node IP: 10.1.1.8
I0713 08:14:12.903012       1 server_others.go:140] Detected node IP 10.1.1.8
W0713 08:14:12.903174       1 server_others.go:598] Unknown proxy mode "", assuming iptables proxy
I0713 08:14:12.976009       1 server_others.go:206] kube-proxy running in dual-stack mode, IPv4-primary
I0713 08:14:12.976279       1 server_others.go:212] Using iptables Proxier.
I0713 08:14:12.976329       1 server_others.go:219] creating dualStackProxier for iptables.
```

How to access my pods from outside?

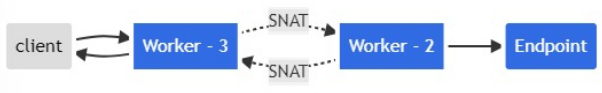
NodePort
Exposing pods to outside (NS traffic)



- The **default port number** range that can be assigned in the worker node is from **30000-32767**

- The nodeport will publish the **nodeport service in all the hosts** including master nodes, regardless whether the pods is inside the host or not

- nodeport will also replace source IP Address with its own IP address

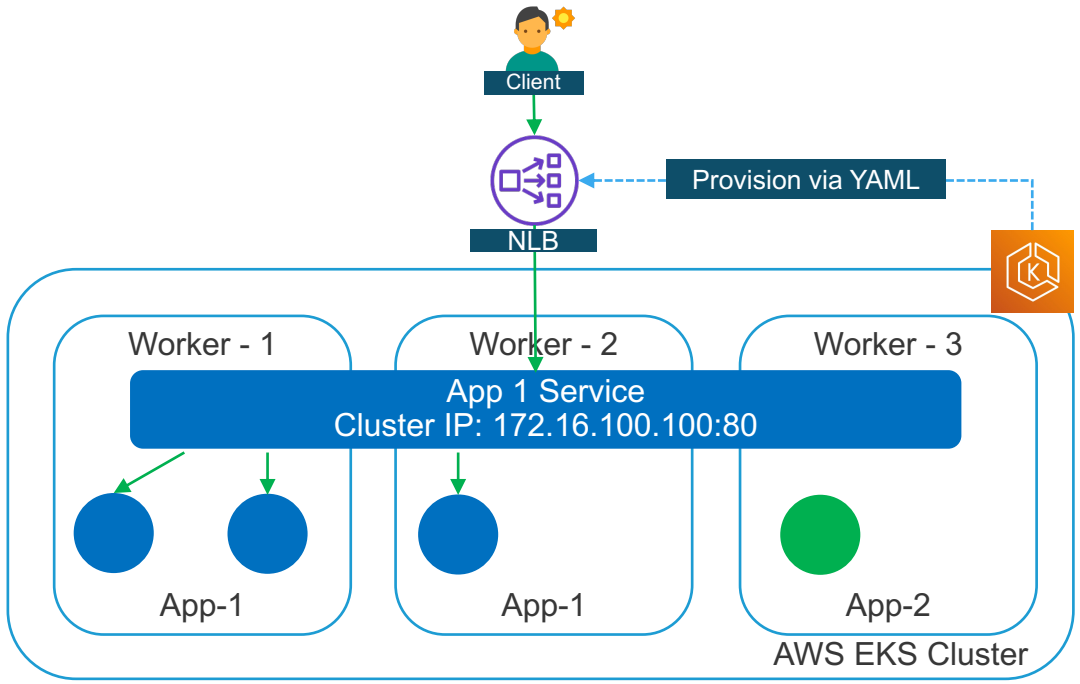


K8s Concept - LoadBalancer

LOAD-BALANCER TYPE IN PUBLIC CLOUD

How to use cloud native load balancer?

Combine both ClusterIP and NodePort




```
apiVersion: v1
kind: Service
metadata:
  name: nlb-sample-service
  namespace: nlb-sample-app
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-
type: external
    service.beta.kubernetes.io/aws-load-balancer-
nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-
scheme: internet-facing
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

Lab Time (Lab 1.6- 1.9)

50 mins

K8s Concept

SUMMARY CNI VS KUBE-PROXY

Summary	CNI	Kube-Proxy
High Level Objectives	To create overlay networks	To intercept traffic and redirect it to correct pods
Pods provisioned on	All worker and master nodes	All worker and master nodes
Specific functions	• Configure the network • Managing IP address allocation within the clusters • Maintain cluster network connectivity	• Route the traffic from pods to pods (ClusterIP) • Route the traffic from outside to pods (Nodeport) • Load-balancing capability for services
Issues if the function is not working	• Pods unable to get IP address • Communication between nodes might have issues	• Pods will have issue connecting to service • External will have issue reaching pods

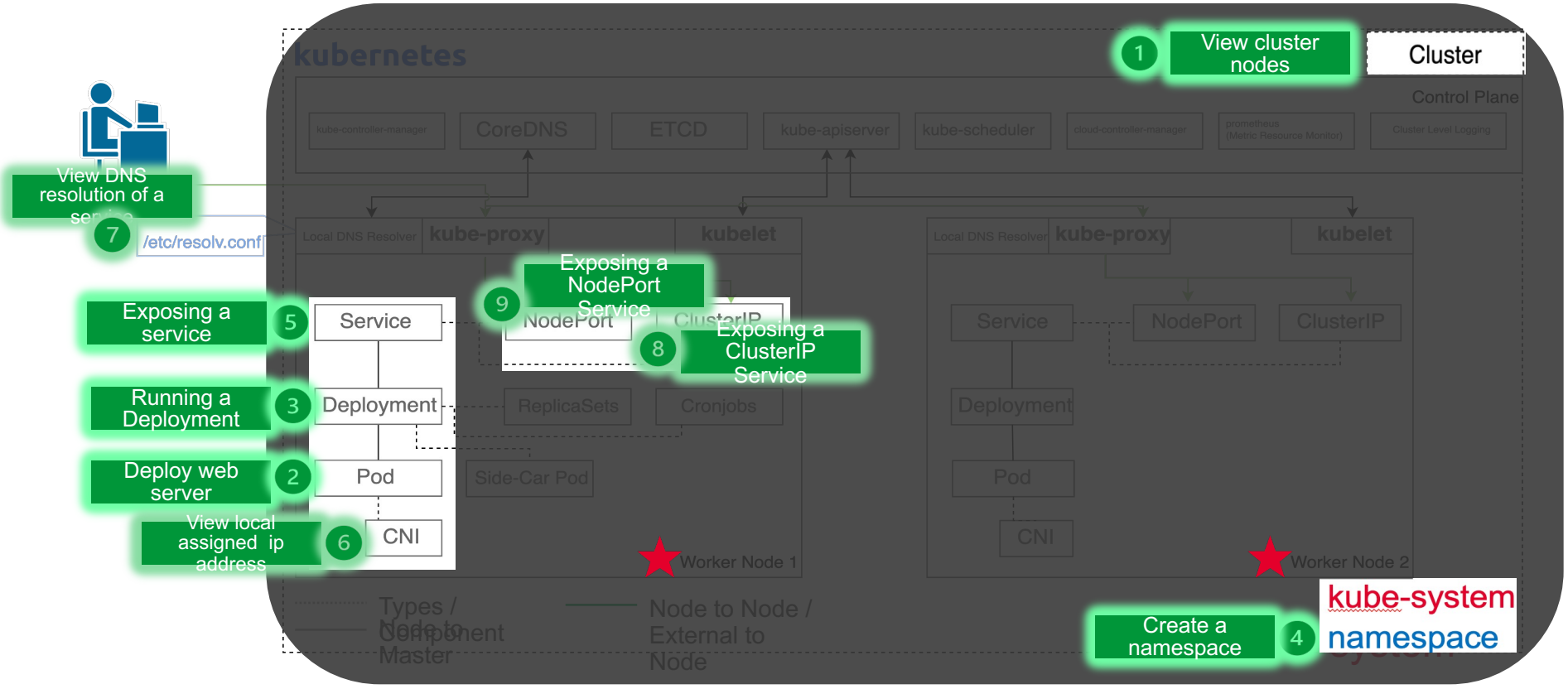
K8s Concept

SUMMARY CLUSTER IP, NODEPORT, LOAD-BALANCER

Summary	Cluster IP	NodePort	Load-Balancer
High Level Objectives	To help connect pods to pods (internal)	To publish the port to be accessed externally	To publish the port to be accessed externally either in public VPC/VNET or private VPC/VNET
Components	Kube-proxy & CNI	Kube-proxy & CNI	Kube-proxy & CNI & Public Cloud NLB
Advantages (Plus Points)	• Simple way to publish the pods INTERNALLY within the cluster across the nodes • Built-in Load-balancing capability to the pods • Can be used for troubleshooting purpose	• Simple way to publish the pods EXTERNALLY outside of the cluster • The pods will be accessible within all nodes in the cluster • Built-in Load-balancing capability to the pods	• Automatically provisioned NLB that will be accessible from the outside • The NLB will load-balance the traffic directly to the pods (more efficient) • The port number can be customed and not restricted to ports 30000–32767
Drawbacks (Negative Points)	• -	• No L7 routing capabilities, traffic will be passed directly to the backend • Require another Load-Balancer to LB between nodes. Otherwise, imbalance traffic might happen • It can only use ports 30000–32767 • If worker nodes IP address change, require changes in the surrounding environment	• No L7 routing capabilities, traffic will be passed directly to the backend • Additional price for the LB provisioned • Every service will need additional LB, it can be expensive if have many services • Only available on cloud environment

Quick Recap

1. **K8s Concepts**
   - Cluster and Nodes
   - Pods and Containers
   - Deployments
   - Service
2. **K8s Networking**
   - Within Cluster
   - Outside Cluster



Section 2 Quiz Time

10 mins

Breaktime

15 mins