

# Kubernetes Intro

# Docker limitation

- ▶ Docker run single instance on the single host.
- ▶ Scaling, load balancing and clustering is not available
- ▶ Container across cluster limitation

Multiple solution available such as Mesosphere, Docker Swarn and Kubernetes.

# Need of Container Orchestration Engine

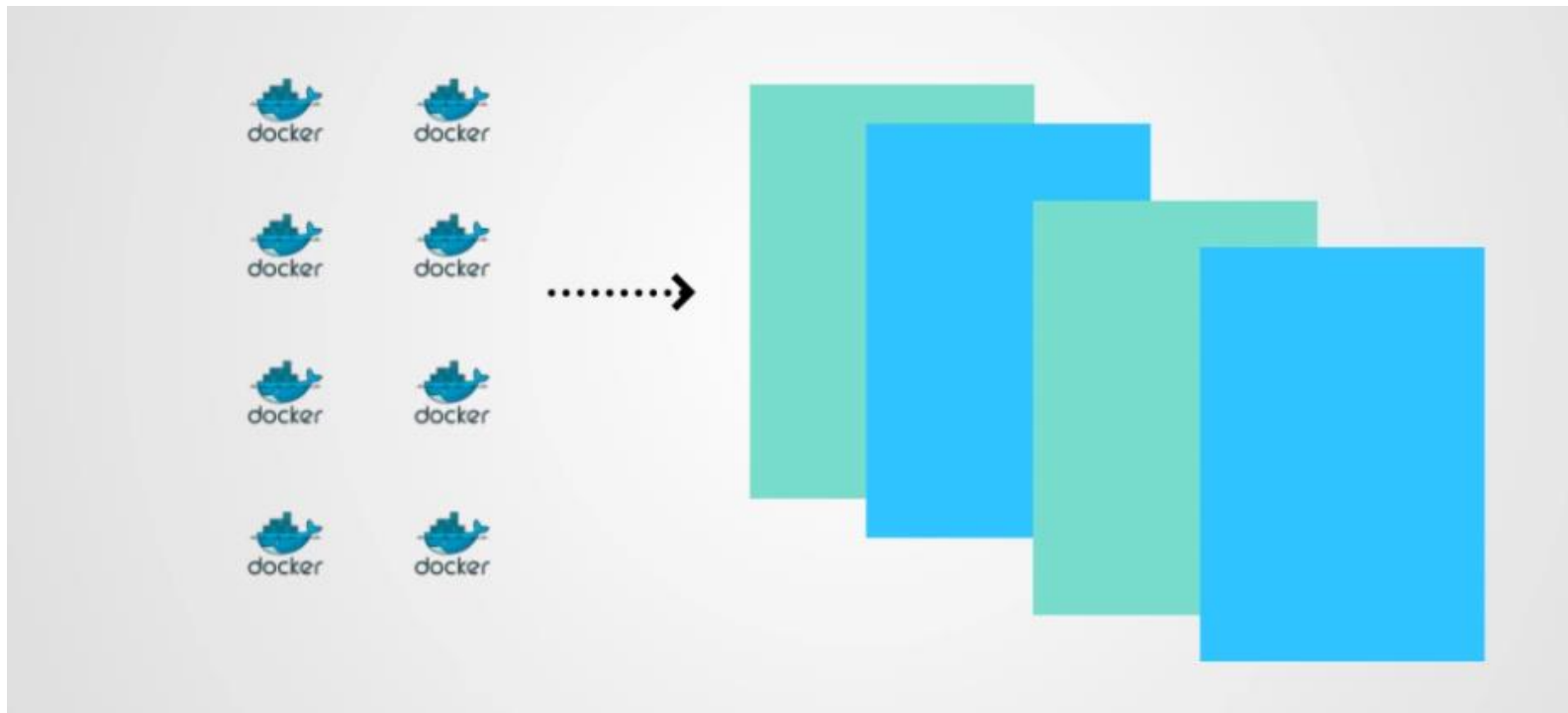
- ▶ Why we need Orchestration Engine
- ▶ What is kubernetes
- ▶ Comparison with similar product

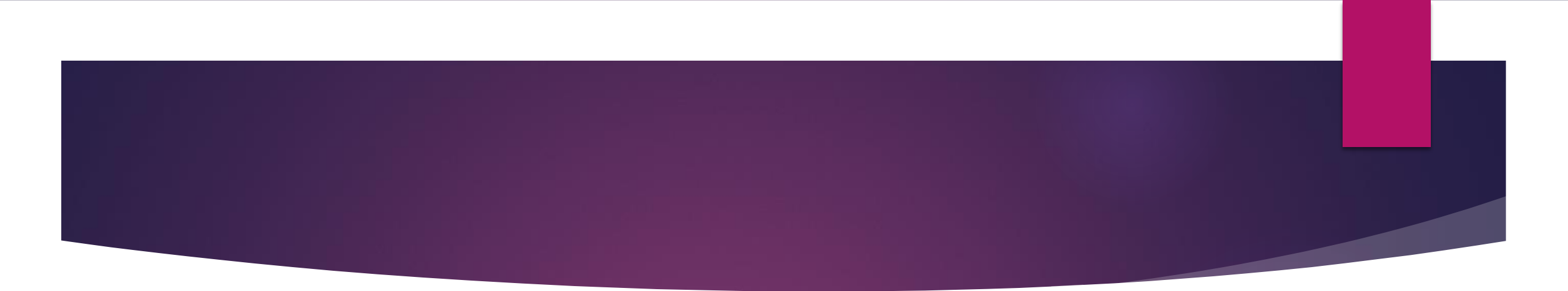
# Why we need??

- ▶ Docker is limited to one single host.
- ▶ When we need to deploy on large scale
- ▶ When we think about high availability, replication
- ▶ Docker as a production ready solution
- ▶ When we talk about application stack, micro services docker compose is the solution, but this is also limited to one single host

Only solution: Container orchestration, what are the solution

# Kubernetes



- 
- ▶ We have pool of servers, where we need to run dockers.
  - ▶ If we don't have orchestration engine, it would be difficult, where to schedule?
  - ▶ How to do load balancing. Manually tracking everything 😞
  - ▶ Without any kind of solution, if we plan to use docker we need to perform scheduling ,management manually...



••

- ▶ Rather than sending the request to each node, we are submitting our request to container orchestrator engine.
- ▶ This engine will take care scheduling the resource, will take care of resource utilization..
- ▶ Take a pool of server, create an logical entity, container orchestrator engine take care of logical entity.

# Comparison of COEs

- ▶ Kubernetes: Robust, most features, its by google.
- ▶ Docker swarm :Provided by Docker. Easy to deploy. Seamless with docker cli.
- ▶ Mesos: Apace project, Mesos is not purely COEs, this is orchestrator. It has concept of datacenter os, so it again create a pool server.
  - ▶ Hadoop job
  - ▶ Batch job
  - ▶ Docker job

Everything require a framework. For docker this require marathon



# Key features of COE

- ▶ Clustering
- ▶ Scheduling
- ▶ Container as service such as cloud providers gives IAAS
- ▶ Dynamic provision, horizontal scaling
- ▶ Load balancing option
- ▶ Fault tolerance
- ▶ Release strategy, rolling update..

# Why kubernetes

- ▶ Created by google, based on their leaning
- ▶ Its based on Borg
- ▶ Google now given to Open source ,community based.
- ▶ Lot of learning from Google.
- ▶ Amount of features
- ▶ It can use for Runc ,rocket
- ▶ Redhat OSE completely based on kubernetes.

# Kubernetes history

- ▶ This is developed by Google
- ▶ Borg was the product name, developed by Google internally
- ▶ Used across multiple product solution.
- ▶ Borg which was developed at Google to manage both long running processes and batch jobs, which was earlier handled by separate systems.
- ▶ Google opensource kubernetes for community. Now this is maintain by opensource communities.

# What is Kubernetes??

- ▶ Kubernetes is opensource container management tool.
- ▶ Kubernetes has the capability of automating deployment, scaling of the application and operations of application containers across cluster
- ▶ Capable of creating container centric infrastructure
- ▶ Kubernetes can run application on clusters of physical and virtual machine.
- ▶ It helps in moving from host-centric infrastructure to container centric infrastructure.
- ▶ Kubernetes is an orchestration framework, which deploy container in unit call pods

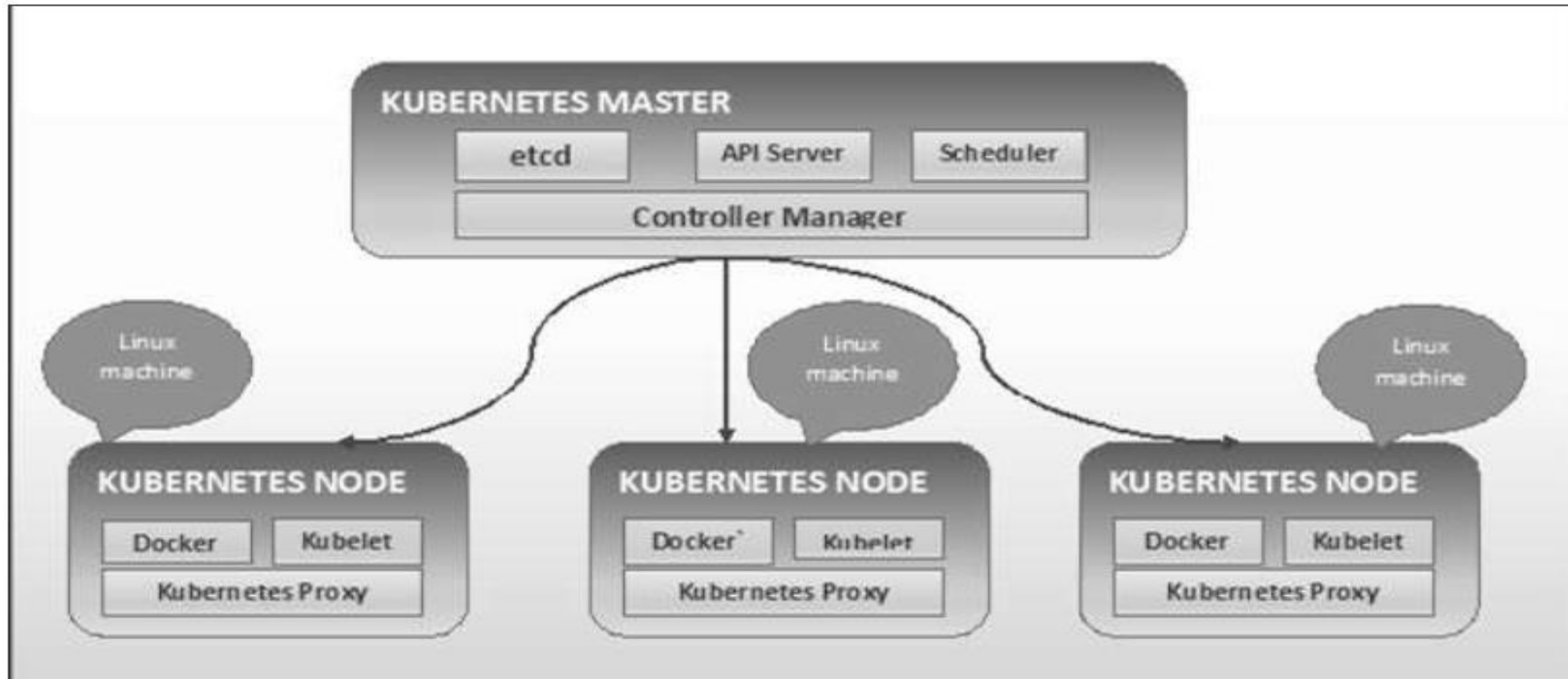
# Kubernetes architecture

- ▶ Kubernetes has a cluster architecture
- ▶ It has client-server [master-node] architecture
- ▶ Master would be on one linux vm with following components
  - ▶ Etcd
  - ▶ Api Server
  - ▶ Scheduler
  - ▶ Controller Manager

...

- ▶ Node would be running on another linux vm
- ▶ Kubernetes can have multiple node
- ▶ Following are the node components:
  - ▶ Kubelet service
  - ▶ Docker/Container run time
  - ▶ Kubernetes proxy

# Kubernetes Architecture





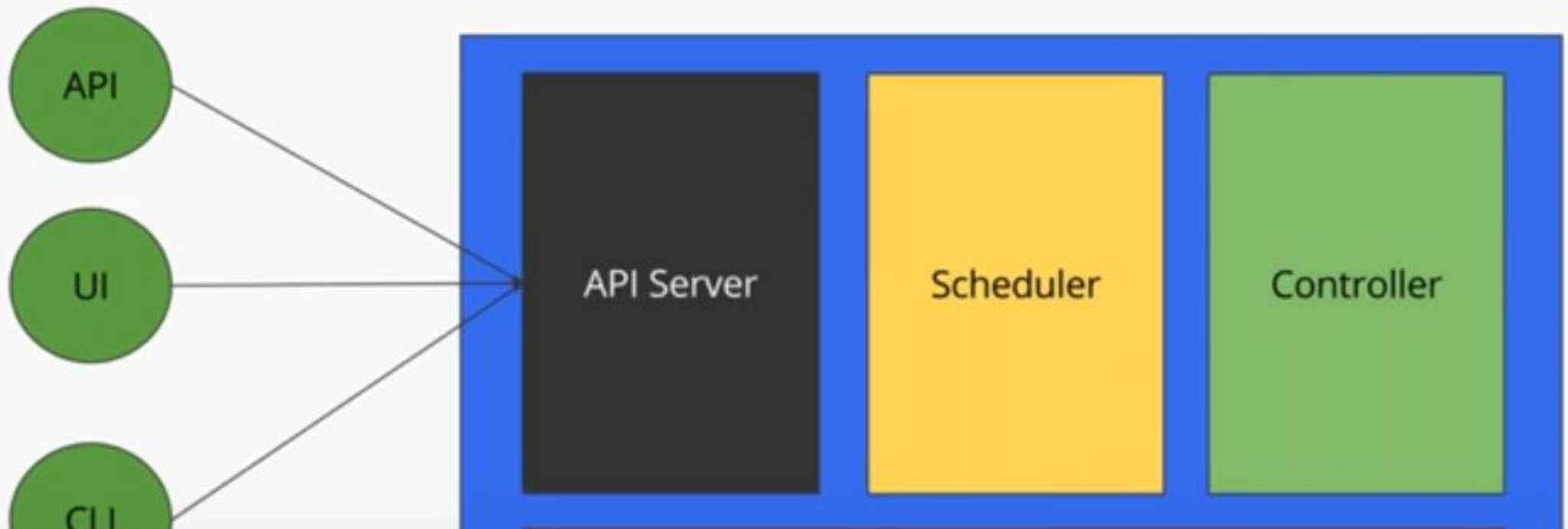
What next?

Lets understand each component in details



# Master components

## Kubernetes Master



# Kube api-server

- ▶ The Kubernetes API server validates and configures data for the api objects which include pods, services, replicationcontrollers, and others.
- ▶ The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.
- ▶ The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.
- ▶ It can scale horizontally

# Master Component: etcd

- ▶ Etcd originated from two ideas Etcd= etc+d
- ▶ etc directory in linux stores the configuration file
- ▶ D stands for distributed system.
- ▶ Hence a “d”istributed “etc” is “etcd”
- ▶ Stores metadata in a consistent and fault-tolerant way.



...

- ▶ Kubernetes stores configuration data into etcd
- ▶ This used for service discovery and cluster management
- ▶ Kubernetes persist cluster state into etcd.
- ▶ Master always keep monitoring the etcd and roll out critical configuration changes.

# Other solution

## ▶ Zookeeper

- ▶ Zookeeper solves the same problem as etcd distributed system coordination and metadata storage.
- ▶ Zookeeper has its own custom jute RPC protocol.

## ▶ Consul

- ▶ Consul bills itself an end-to-end service discovery framework.
- ▶ For complete service discovery Consul is the best solution with kubernetes.

# kube-scheduler

- ▶ Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.
- ▶ Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines

# kube-controller-manager

- ▶ Control plane component that runs controller processes.
- ▶ Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.
- ▶ Some types of these controllers are:
  - ▶ Node controller: Responsible for noticing and responding when nodes go down.
  - ▶ Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
  - ▶ Endpoints controller: Populates the Endpoints object (that is, joins Services & Pods).
  - ▶ Service Account & Token controllers: Create default accounts and API access tokens for new namespaces

# kubelet

- ▶ An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.
- ▶ The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes



# Kube-proxy

- ▶ kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.
- ▶ kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.



...

Kubernetes we use kubectl as cli

Kubernetes bit complex, lot of features.

- Lot of objects
- API format
- Understand the api stack
- Understand object stack
- AKMS format

# Key features of k8s

**FEATURES OF A K8S**

	<b>Feature</b>	<b>Concept</b>
<b>clustering</b>	Colocation	Pods
<b>scheduling</b>	Scaling/Fault Tolerance	replication controllers, replica sets
<b>scaling</b>	Load Balancing	Services
<b>load balancing</b>	App Deployment, Rollbacks	deployments, rolling-updates
<b>fault tolerance</b>	Stateful Apps	stateful sets/ pet sets
<b>app deployment</b>	configs, secrets	config sets, secrets
	app health checks	liveness, readiness
	logs and storage	
	monitoring	

5

# Setting up Kubernetes using kubeadm

- ▶ kubeadm: the command to bootstrap the cluster.
- ▶ kubelet: the component that runs on all of the machines in your cluster and does things like starting pods and containers.
- ▶ kubectl: the command line util to talk to your cluster.

# Setting up k8s cluster using kubeadm

- ▶ `kubeadm init <args>`
- ▶ `kubeadm init` first runs a series of prechecks to ensure that the machine is ready to run Kubernetes. These prechecks expose warnings and exit on errors. `kubeadm init` then downloads and installs the cluster control plane components
- ▶ Choose a Pod network add-on, and verify whether it requires any arguments to be passed to `kubeadm init`. Depending on which third-party provider you choose, you might need to set the `--pod-network-cidr` to a provider-specific value

```
.....
root@ubuntu20:~/test# kubectl init --pod-network-cidr=10.244.0.0/16
[init] Using Kubernetes version: v1.25.0
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubectl config images pull'
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [kubernetes kubernetes.default kubernetes.default.
fault.svc.cluster.local ubuntu20] and IPs [10.96.0.1 157.245.108.248]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [localhost ubuntu20] and IPs [157.245.108.248 12
[certs] Generating "etcd/peer" certificate and key
```



••

To start using your cluster, you need to run the following as a regular user:

```
mkdir -p $HOME/.kube
```

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

# Check the nodes status

```
root@ubuntu20:~/test# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
ubuntu20	NotReady	control-plane	2m2s	v1.25.0

```
root@ubuntu20:~/test#
```



# Setup network

- ▶ You must deploy a Container Network Interface (CNI) based Pod network add-on so that your Pods can communicate with each other.
- ▶ Cluster DNS (CoreDNS) will not start up before a network is installed.
- ▶ Flannel or calico can be used for the same
- ▶ `kubectl apply -f`  
<https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

```
root@ubuntu20:~/test# kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
coredns-565d847f94-nm6hr	0/1	Pending	0	7m53s
coredns-565d847f94-vtb6k	0/1	Pending	0	7m53s
etcd-ubuntu20	1/1	Running	0	8m6s
kube-apiserver-ubuntu20	1/1	Running	0	8m6s
kube-controller-manager-ubuntu20	1/1	Running	0	8m6s
kube-proxy-vkm79	1/1	Running	0	7m53s
kube-scheduler-ubuntu20	1/1	Running	0	8m6s

```
root@ubuntu20:~/test# kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
coredns-565d847f94-nm6hr	0/1	ContainerCreating	0	7m58s
coredns-565d847f94-vtb6k	0/1	ContainerCreating	0	7m58s
etcd-ubuntu20	1/1	Running	0	8m11s
kube-apiserver-ubuntu20	1/1	Running	0	8m11s
kube-controller-manager-ubuntu20	1/1	Running	0	8m11s
kube-proxy-vkm79	1/1	Running	0	7m58s
kube-scheduler-ubuntu20	1/1	Running	0	8m11s

```
root@ubuntu20:~/test# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
ubuntu20	Ready	control-plane	8m19s	v1.25.0

```
root@ubuntu20:~/test# kubectl get pods -n kube-system
```