kubeadm

❖ KUBEADM:

- Kubeadm is a tool built to provide kubeadm init and kubeadm join as best-practice "fast paths" for creating Kubernetes clusters.
- kubeadm performs the actions necessary to get a minimum viable cluster up and running.
- By design, it cares only about bootstrapping, not about provisioning machines (underlying worker and master nodes).
- Kubeadm also serves as a building block for higher-level and more tailored tooling.

➢ KUBEADM'S FEATURES:

- Common use cases for Kubeadm include testing, creating baselines for more advanced K8s deployments, and providing new K8s users a simple starting point for cluster configuration.
- The specific features that make kubeadm useful in those applications are:

QUICK MINIMUM VIABLE CLUSTER CREATION:

Kubeadm is designed to have all the components you need in one place in one cluster regardless of where you are running them.

PORTABLE:

Kubeadm can be used to set up a cluster anywhere whether it's your laptop, a Raspberry Pi, or public cloud infrastructure.

LOCAL DEVELOPMENT:

As Kubeadm creates clusters with minimal dependencies and quickly, it's an ideal candidate for creating disposable clusters on local machines for development and testing needs.

BUILDING BLOCK FOR OTHER TOOLS:

Kubeadm is not just a K8s installer. It also serves as a building block for other tools like Kubespray.

➢ KUBEADM INSTALLATION:

KUBEADM-PREREQUISITES:

- A compatible Linux host. (Debian and Red Hat)
- 2 GB or more of RAM per machine
- 2 CPUs or more.
- Full network connectivity between all machines in thecluster
- Unique hostname, MAC address, and product_uuid forevery node.
- Certain ports are open on your machines.
- MUST disable swap

LAB-SETUP:

Master	Hostname: Master (10.10.10.100)	2GB Ram, 2vcpus
Worker1	Hostname: Node1 (10.10.10.101)	1GB Ram, 1vcpus
Worker2	Hostname: Node2 (10.10.10.102)	1GB Ram, 1vcpus

STEP 1: Seting up Hostname on each vm based on LAB-SETUP

```
#hostname Master
#vim /etc/hostname
    Master
#vim /etc/hosts
    Master-IP     Master
    Node1-IP      Node1
    Node2-IP      Node2
```

FROM Node1:

```
#hostname Node1
#vim /etc/hostname
    Node1
#vim /etc/hosts
    Node1-IP      Node1
    Master-IP     Master
#hostname Node2
```

FROM Node2

```
#vim /etc/hostname
      Node2
#vim /etc/hosts
       Node2-IP      Node2
       Master-IP     Master
```

Update hostname run: #bash

STEP 2: Turnoff swap on all Master & Worker Nodes:

```
#swapoff -a
#vim /etc/fstb
     Comment a swap file system (#)
#systemctl daemon-reload
```

STEP 3: Disable SE-Linux firewalls on all master & Worker Nodes:

```
#setenforce 0
#vim /etc/selinux/config
  SELINUX=disabled
#reboot
```

INSTALLING A CONTAINER RUN TIME ON ALL:

To run containers in Pods, Kubernetes uses a container runtime.

STEP 4: Set up the repository

```
#yum install -y yum-utils
```

```
#yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
```

STEP 5: Install Docker Engine

```
#yum install docker-ce -y
```

STEP 6: Start and Enable docker service

```
#systemctl start docker
#systemctl enable docker
#systemctl status docker
#docker version
```

INSTALLING KUBEADM, KUBELET AND KUBECTL:

- You will install these packages on all of your machines:

  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.

STEP 7: Setup a repository:

```
#cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
   [kubernetes]
   name=Kubernetes
   baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-
   \$basearch
   enabled=1
   gpgcheck=1
   gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
   exclude=kubelet kubeadm kubectl
   EOF
```

STEP 8: Installing and Enable Kubelet:

```
#yum install -y kubelet kubeadm kubectl --disableexcludes=Kubernetes
#systemctl start kubelet
#systemctl enable --now kubelet
```

CREATING A CLUSTER WITH KUBEADM:

- **The kubeadm tool is good if you need:**
  - A simple way for you to try out Kubernetes, possibly for the first time.
  - A way for existing users to automate setting up a cluster and test their application.
  - A building block in other ecosystem and/or installer tools with a larger scope.

Before You Begin:

- One or more machines running a deb/rpm-compatible Linux OS; for example: Ubuntu or CentOS.
- 2 GiB or more of RAM per machine--any less leaves little room for your apps.
- At least 2 CPUs on the machine that you use as a control-plane node.
- Full network connectivity among all machines in the cluster. You can use either a public or a private network.

INITIALIZE KUBERNETES CLUSTER (FROM MASTER):

- The control-plane node is the machine where the control plane components run, including etcd (the cluster database) and the API Server (which the kubectl command line tool communicates with).

To initialize the control-plane node run:

```
#kubeadm init  --apiserver-advertise-address=10.10.10.100
```

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

```
$mkdir -p $HOME/.kube
$sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Alternatively, if you are the root user, you can run:#export

```
#KUBECONFIG=/etc/kubernetes/admin.conf
#kubectl get pods --all-namespaces
#kubectl get nodes
```

JOINING NODES:

To add new nodes to your cluster do the following for each machine:

You can join any number of worker nodes by running the following on each as root:

```
#kubeadm join 10.10.10.10:6443 --token tpj9f5.ikl2z77ufimmwos3 \
    --discovery-token-ca-cert-hash
    sha256:dd6c10b0bb9efa3062017926806e77173baf5b80ee1ee7486867ddc
```

If you do not have the token, you can get it by running the following command on the control-plane node:

```
#kubeadm token list
```

NOTE: By default, tokens expire after 24 hours. If you are joining a node to the cluster after the current token has expired, you can create a new token by running the following command on the control-plane node:

```
#kubeadm token create
```

After Joining nodes from the Master:

```
#kubectl get nodes
#kubectl get pods --all-namespaces
```

TROUBLE SHOOTING:

Before removing the node, reset the state installed bykubeadm:

```
#kubeadm reset    [From worker node]
```

Remove the Nodes [From Master]

```
#kubectl drain <node name> --delete-local-data --force --ignore-daemonsets
```

Now remove the node:

```
#kubectl delete node <node name>
```

To get a hash again to join nodes:

```
#kubeadm token create --print-join-command
```

CLEAN UP THE CONTROL PLANE:

Use **kubeadm reset** on the control plane host to trigger a best-effort clean up

```
#kubeadm reset
```