Kubernetes installation:

Control plane

| **Protocol** | **Direction** | **Port Range** | **Purpose** | **Used By** |
| --- | --- | --- | --- | --- |
| TCP | Inbound | 6443 | Kubernetes API server | All |
| TCP | Inbound | 2379-2380 | etcd server client API | kube-apiserver, etcd |
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 10259 | kube-scheduler | Self |
| TCP | Inbound | 10257 | kube-controller-manager | Self |

| **Protocol** | **Direction** | **Port Range** | **Purpose** | **Used By** |
| --- | --- | --- | --- | --- |
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 30000-32767 | NodePort Services† | All |

Worker node(s)

🡪 Swap disabled. You **MUST** disable swap in order for the kubelet to work properly

🡪 sudo swapoff -a

🡪 sudo vi /etc/hosts --🡪

65.0.176.183 master

3.110.54.18 worker1

3.110.204.55 worker2

🡪sudo hostnamectl set-hostname master (for nodes also)

**Containerd (both master and worker nodes)**

This section contains the necessary steps to use containerd as CRI runtime.

Use the following commands to install Containerd on your system:

Install and configure prerequisites:

cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf

overlay

br\_netfilter

EOF

sudo modprobe overlay

sudo modprobe br\_netfilter

*# Setup required sysctl params, these persist across reboots.*

cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

EOF

*# Apply sysctl params without reboot*

sudo sysctl –system

**Install containerd:**

1.Install the containerd.io package from the official Docker repositories. Instructions for setting up the Docker repository for your respective Linux distribution and installing the containerd.io package can be found at [Install Docker Engine](https://docs.docker.com/engine/install/#server).

-🡪 sudo apt-get update

🡪 sudo apt-get install containerd

2. Configure containerd:

sudo mkdir -p /etc/containerd

containerd config default | sudo tee /etc/containerd/config.toml

3. Restart containerd:

sudo systemctl restart containerd

script to install and configure containerd:

install.sh

cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf

overlay

br\_netfilter

EOF

sudo modprobe overlay

sudo modprobe br\_netfilter

*# Setup required sysctl params, these persist across reboots.*

cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

EOF

*# Apply sysctl params without reboot*

sudo sysctl --system

sudo apt-get update

sudo apt-get install containerd -y

sudo mkdir -p /etc/containerd

containerd config default | sudo tee /etc/containerd/config.toml

sudo systemctl restart containerd

service containerd status

**Installing kubeadm, kubelet and kubectl(both master and workernodes)**

Update the apt package index and install packages needed to use the Kubernetes apt repository:

sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl

Download the Google Cloud public signing key:

sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

Add the Kubernetes apt repository:

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

Update apt package index, install kubelet, kubeadm and kubectl, and pin their version:

sudo apt-get update

sudo apt-get install -y kubelet=1.21.0-00 kubeadm=1.21.0-00 kubectl=1.21.0-00

sudo apt-mark hold kubelet kubeadm kubectl

ref: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#check-required-ports>

[kubeadm init](https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-init) to bootstrap a Kubernetes control-plane node

initialize cluster:

-🡪sudo kubeadm init

Access master: (thereis only one entry-point i.e api server)

Using kubeconfig file (contains api address, certificate and key)

Kubectl

🡪 kubectl get node (get error)

🡪 sudo kubectl get node –kubeconfig /etc/kubernates/admin.conf

Create admin.conf as environment variable

🡪sudo -i

-🡪 export KUBECONFIG=/etc/kubernetes/admin.conf

🡪 kubectl get node

🡪exit

ls ~/.kube

mkdir ~/.kube

sudo cp -i /etc/kubernetes/admin.conf ~/.kube/config

ls -l ~/.kube/config

sudo chown ubuntu:ubuntu /home/ubuntu/.kube/config

ls -l ~/.kube/config

namespace:

🡪 kubectl get namespace

Configuring networking: weave net

Ref : <https://www.weave.works/docs/net/latest/kubernetes/kube-addon/>

<https://www.weave.works/docs/net/latest/tasks/ipam/configuring-weave/>

🡪 wget "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')" -O weave.yaml

🡪 vi weave.yaml

containers:

- name: weave

command:

- /home/weave/launch.sh

- --ipalloc-range=100.32.0.0/12 --- add this line

🡪 kubectl apply -f weave.yaml

🡪 kubectl get node --- now master running

🡪 kubectl get pod -n kube-system

🡪 kubectl get pod -n kube-system -o wide --- checking ports

Add worker nodes:

🡪 sudo swapoff -a ( worker nodes)

🡪 kubeadm token create –print-join-command (master)

🡪 kubeadm join 172.31.2.216:6443 --token sld7qr.j8uwiu6apysd2inc --discovery-token-ca-cert-hash sha256:8825bcfc2a69f67b589b4cc4b6b155d4bb0a8c52549f8eb76f3bf72dcf5c0057 (worker node)

-> kubectl logs weave-net-hm9cs -n kube-system -c weave (checking logs and add inbound rule:6784)

🡪 kubectl get pod -o wide | grep weave-net

🡪 kubectl exec -n kube-system weave-net-hm9cs -c weave -- /home/weave/weave --local status

Deploy application:

kubectl run test --image=nginx

kubectl get pod

kubectl get pod -w

kubectl get pod -o wide

kubectl run test2 --image=nginx

kubectl get pod -o wide

create ngnix deployement:

ref: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

🡪 vi nginx-deployment.yaml

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: nginx-deployment

**labels**:

**app**: nginx

**spec**:

**replicas**: 2

**selector**:

**matchLabels**:

**app**: nginx

**template**:

**metadata**:

**labels**:

**app**: nginx

**spec**:

**containers**:

- **name**: nginx

**image**: nginx

**ports**:

- **containerPort**: 80

🡪 kubectl apply -f nginx.yaml

Note: Template: template has its own meta data and spec section.

-🡪 this configuration (template configuration) applies to pod.

🡪 blue print of pod

🡪 kubectl get pod -o wide

🡪 kubectl get deployment

Create nginx service:

Ref: <https://kubernetes.io/docs/concepts/services-networking/service/>

🡪 vi nginx-service.yaml

**apiVersion**: v1

**kind**: Service

**metadata**:

**name**: nginx-service

**spec**:

**selector**:

**app**: nginx

**ports**:

- **protocol**: TCP

**port**: 8080

**targetPort**: 80

🡪 kubectl apply -f nginx-service.yaml

🡪 kubectl get svc -o wide

🡪kubectl describe svc service-name (detail information about svc)

Note:

How does service forward requests to ip address:

🡪 service is a not an application or process running on nodes.

🡪 Is actually just virtual IP address accessible through out the cluster.

Request 🡪 service(1000.0.9.1)🡪 kube proxy 🡪 pod

Kube proxy:

🡪 kube proxy forwards the request

🡪responsible for maintaining list of services ips and corresponding pod ips.

🡪 kubectl get all -- gives all the components

Scaling & recording kubectl commands: (instead of using deployment configuration file)

🡪 kubectl scale deployment nginx-deployment –replicas=4

🡪 kubectl get pod

🡪 kubectl scale deployment nginx-deployment –replicas=3 (scale down)

🡪 kubectl scale deployment nginx-deployment --replicas=5 --record=true (record the changes in deployement)

🡪 kubectl rollout history deployment nginx-deployment (got history of that cahnges)

Test nginx app accessibility:

🡪 Kubectl run test-nginx-svc –image=nginx (created pod)

🡪 kubectl get pod

🡪 kubectl exec -it name\_pod – bash (inter in to and execute commands).

🡪curl <http://servicename> or sevice ip: 8080 (check service accessible or not)

Note: api server assign default cidr block(service ip) to service.

Created configuration files for different components using kubectl commonds

Service configuration file

🡪 kubectl create service clusterip new-cidr --tcp=8080:80 --dry-run=client -o yaml > my-svc.yaml

Deployment configuration file:

🡪 kubectl create deployment my-deployment –image=nginx –port=8080 –replicas=3 –dry-run -o yaml > my-deploy.yaml

Pod configuration file:

🡪 kubectl run my-pod --image=nginx --dry-run=client -o yaml > my-pod.yaml

Application accessible through outside cluster

Three ways: 1. Node port service type

2. Load balancer service Type

Node port service type:

[NodePort](https://kubernetes.io/docs/concepts/services-networking/service/#type-nodeport): Exposes the Service on each Node's IP at a static port (the NodePort). A ClusterIP Service, to which the NodePort Service routes, is automatically created. You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>.

🡪 kubectl delete svc nginx-service

🡪 vi nginx-service.yaml ( add type: NodePort, and nodePort: 30000)

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

type: NodePort

selector:

app: nginx

ports:

- protocol: TCP

port: 8080

targetPort: 80

nodePort: 30000

Load Balancer Service type:

* [LoadBalancer](https://kubernetes.io/docs/concepts/services-networking/service/#loadbalancer): Exposes the Service externally using a cloud provider's load balancer. NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created.

🡪 kubectl delete svc nginx-service

🡪 vi nginx-service.yaml ( add type: loadbalancer)

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

type: LoadBalancer

selector:

app: nginx

ports:

- protocol: TCP

port: 8080

targetPort: 80

nodePort: 30000

🡪 create load balancer and access service through dns name.

Ingress service type:

[Ingress](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.23/#ingress-v1-networking-k8s-io) exposes HTTP and HTTPS routes from outside the cluster to [services](https://kubernetes.io/docs/concepts/services-networking/service/) within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

Helm:

🡪 It is a package manager for Kubernetes.

🡪 To package YAML files and distribute them in public and private repos

🡪 so, we reuse the configuration.

Install helm: (to get ingress controller)

curl https://baltocdn.com/helm/signing.asc | sudo apt-key add -

sudo apt-get install apt-transport-https --yes

echo "deb https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list

sudo apt-get update

sudo apt-get install helm

helm version

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

helm ls

kubectl get pods -o wide

kubectl get svc -o wide

🡪 create new load balancer with respective port

K8s ingress component:

🡪 ingress component is like a configuration piece for the ingress controller.

🡪 we define http rules, which the ingress controller fulfills.

Request🡪 load balancer(32675)🡪 ingress controller🡪service

http://ingressload-balancer-622609140.ap-south-1.elb.amazonaws.com