

Cloud Native COMPUTING BOOT Camp

Final Presentation Tasks



January 4, 2023

Bashir AHmed Zeeshan

Candidate-KHI-RH-4

Table of Contents

[**Bootcamp Final Presentation** 2](#_Toc123688349)

[Domain 2](#_Toc123688350)

[Outcome: 2](#_Toc123688351)

[**Key Task Assigned** 2](#_Toc123688352)

[Installation of Kubernetes cluster 2](#_Toc123688353)

[**Multi-Tier Application** 3](#_Toc123688354)

[Prerequisites: 3](#_Toc123688355)

[**Kubernetes Resources** 4](#_Toc123688356)

[**Deployment Steps** 4](#_Toc123688357)

[Backend Layer 4](#_Toc123688358)

[Middle Layer 5](#_Toc123688359)

[Frontend Layer 6](#_Toc123688360)

[Access Deployment 7](#_Toc123688361)

[Testing Environment 8](#_Toc123688362)

[Deployment Procedure 8](#_Toc123688363)

[Ending Remark 9](#_Toc123688364)

[Task number 02 10](#_Toc123688365)

[**Dockerized a webpage & Push over Docker Hub** 10](#_Toc123688366)

[**Containerization of Web Page & expose it** 12](#_Toc123688367)

[**Appendix-A** 13](#_Toc123688368)

[Kubernetes Cluster Installation 13](#_Toc123688369)

[**Appendix-B** 16](#_Toc123688370)

[YAML for Multi-Tier Application 16](#_Toc123688371)

Cloud Native Computing – Boot Camp

Final Presentation Tasks

**Bashir Ahmed Zeeshan**

# **Bootcamp Final Presentation**

As an essential task of Final Presentation of Cloud Native Computing Bootcamp, it is necessary to pick a specific topic inside Container Kubernetes Administration, which is a vast topic in and of itself, but for the purpose of introduction, Workload and Scheduling is chosen.

## Domain

* Container Kubernetes Administration

## Outcome:

By the end of this part, we will be completing different workloads of Kubernetes with complete hands-on practice and troubleshooting part if required.

# **Key Task Assigned**

For the sake of final Presentation of cloud native computing boot camp. We started deployment of multi-tier application and the its services in Kubernetes Cluster

## Installation of Kubernetes cluster

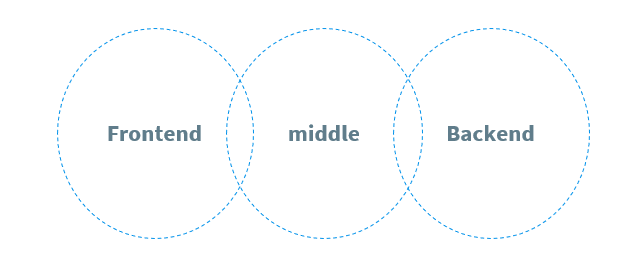
We initiated Kubernetes Cluster deployment on Ubuntu 22.04 with a master node and two worker nodes.

* Master hostname (master-node)
* worker node1 (worker-node1)
* worker node2 (worker-node2)

All details and step by step procedure can be seen Appendix – A

# **Multi-Tier Application**

A multi-tier application is one that is spread over many layers. It divides the operational levels logically. The number of layers varies depending on business and application needs. This program has two layers: the backend and the frontend.



We will utilize Postgres DB as the data storage backend, a Java REST API which serves words read from the database and GO app as the Webserver frontend. These all three apps will be deployed using Kubernetes deployments, with services forwarding requests to them.

## Prerequisites:

Kubernetes Cluster with at least 1 worker node and 2 is good.

Basic understanding of Deployments and Services.

# **Kubernetes Resources**

Using different K8s resources we build a multi-tier web based application

Wordsmith is the demo project shown at DockerCon EU 2017 and 2018.

The demo app runs across three containers:

**\*\*db\*\*** - a Postgres database which stores words

**\*\*words\*\*** - a Java REST API which serves words read from the database

**\*\*web\*\*** - a Go web application which calls the API and builds words into sentences:

YAML of all three deployment and services can be seen Appendix – B

# **Deployment Steps**

## Backend Layer

Create a new file with the following content. This will define a deployment of Postgres DB which will act as a database, a backend layer

Now, create a service that will serve requests to the DB from our middle layer. This service will listen on Port 5432 and forward requests to DB

apiVersion: v1

kind: Service

metadata:

  name: db

  labels:

    app: words-db

spec:

  ports:

    - port: 5432

      targetPort: 5432

      name: db

  selector:

    app: words-db

  clusterIP: None

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: db

  labels:

    app: words-db

spec:

  selector:

    matchLabels:

      app: words-db

  template:

    metadata:

      labels:

        app: words-db

    spec:

      containers:

      - name: db

        image: dockersamples/k8s-wordsmith-db

        ports:

        - containerPort: 5432

          name: db

## Middle Layer

Now we create a new file with the following content. This will define a deployment of GO web based app which will act as a Frontend layer

And also create a service that will serve requests to the DB from our middle layer. This service will listen on Port 8080 and forward requests

**\*\*words\*\*** - a Java REST API which serves words read from the database

apiVersion: v1

kind: Service

metadata:

  name: words

  labels:

    app: words-api

spec:

  ports:

    - port: 8080

      targetPort: 8080

      name: api

  selector:

    app: words-api

  clusterIP: None

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: words

  labels:

    app: words-api

spec:

  replicas: 5

  selector:

    matchLabels:

      app: words-api

  template:

    metadata:

      labels:

        app: words-api

    spec:

      containers:

      - name: words

        image: dockersamples/k8s-wordsmith-api

        ports:

        - containerPort: 8080

          name: api

## Frontend Layer

Now we create a new file with the following content. This will define a deployment of Java Rest API which will act as a middle layer

And also create a service that will serve requests to our frontend layer. This service will listen on Port 8080 and forward requests

**\*\*web\*\*** - a Go web application which calls the API and builds words into sentences:

apiVersion: v1

kind: Service

metadata:

  name: web

  labels:

    app: words-web

spec:

  ports:

    - port: 8081

      targetPort: 80

      name: web

  selector:

    app: words-web

  type: LoadBalancer

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: web

  labels:

    app: words-web

spec:

  selector:

    matchLabels:

      app: words-web

  template:

    metadata:

      labels:

        app: words-web

    spec:

      containers:

      - name: web

        image: dockersamples/k8s-wordsmith-web

        ports:

        - containerPort: 80

          name: words-web

## Access Deployment

To access the frontend, we will create a service of type NodePort listening on Port ranging from 30000 to 32767 (for ease we can fix a port as 31003). This means the Webserver can be accessed on IP=IP-Of-Any-Node and Port=31003.

# Testing Environment

Now, we are all set to create a multi-tier application. Before we proceed, let's check if the cluster has any objects.

Execute the following commands to check pods, deployments and services running in the cluster.

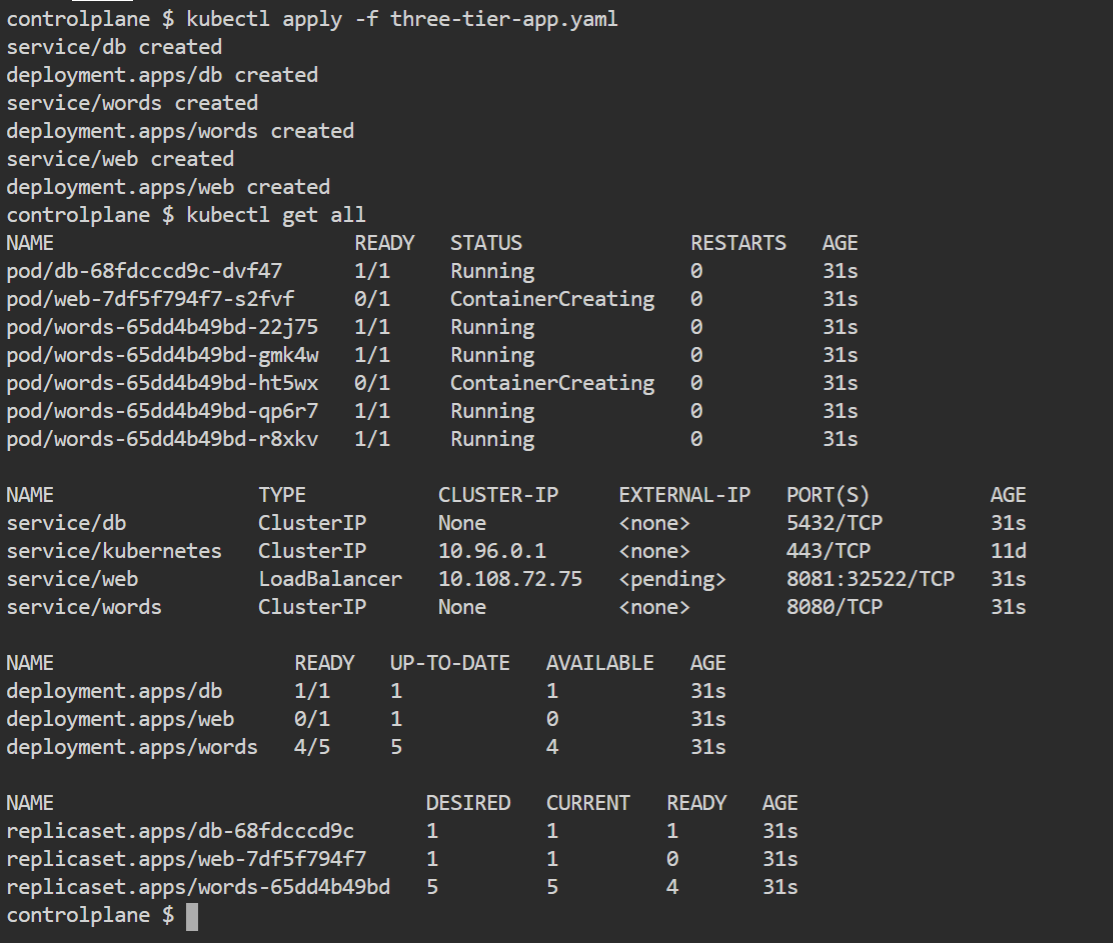
* kubectl get pods
* kubectl get deployments
* kubectl get service

# Deployment Procedure

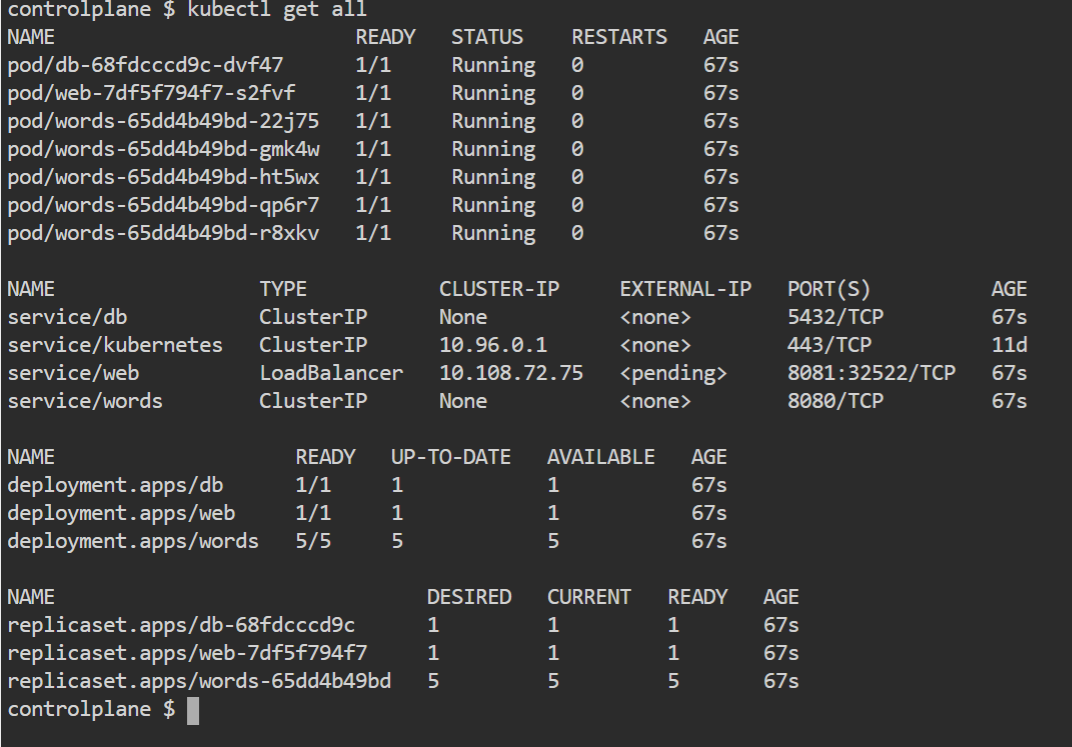
Now, execute the following command one by one from the directory where you have created 4 files to create Postgres DB deployment, service, Java Rest API deployment & services and GO app Web deployment and service.

or If all \*.yaml file have same directory

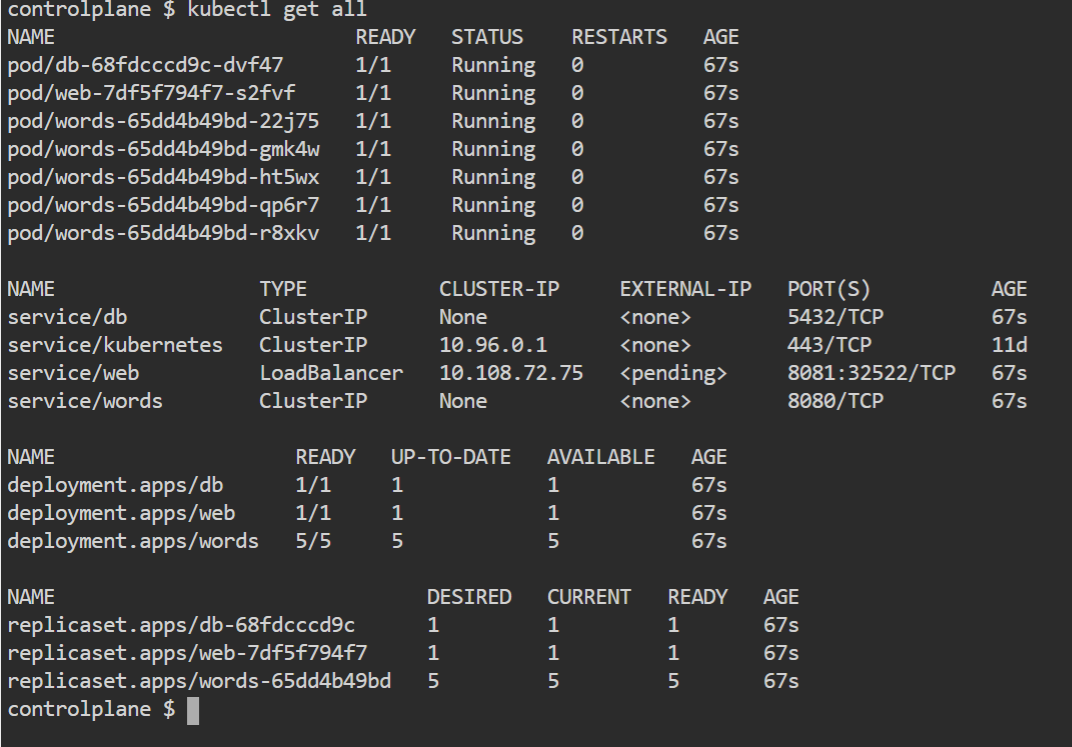
* kubectl create -f .



Now all pods are ready



Now all Kubernetes resources are ready



## Ending Remark

In this task, we created a deployment of the Postgres DB as backend database and service to access the database and for the frontend, we created GO-based Web application deployment and service to access this frontend. We also saw the data getting stored in the backend database.

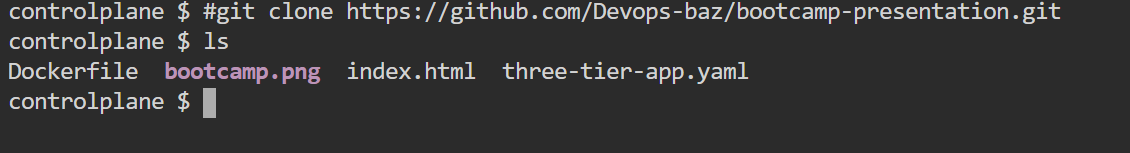


# Task number 02

# **Dockerized a webpage & Push over Docker Hub**

First, I cloned github repo to get all files

git clone https://github.com/Devops-baz/bootcamp-presentation.git



Then I build an image with Dockerfile

docker build -t bootcamp:2022 .

docker images

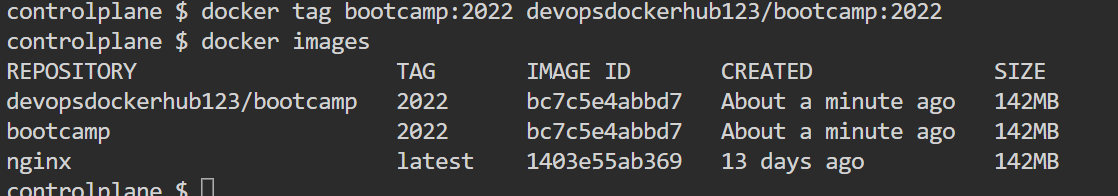
docker login

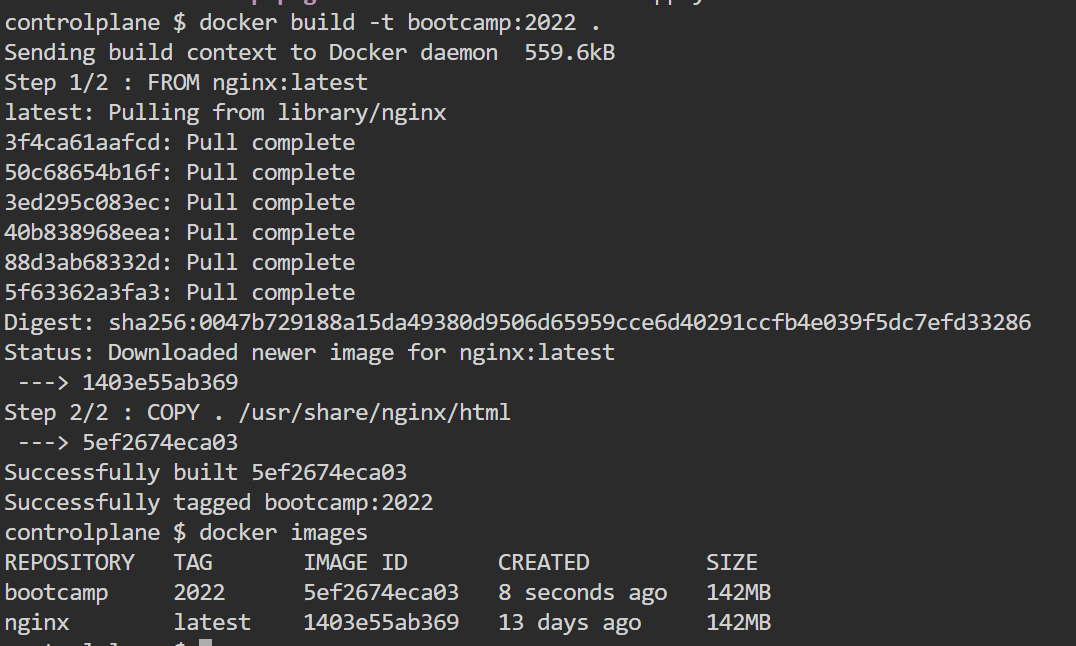
devopsdockerhub123

\*\*\*\*\*\*\*

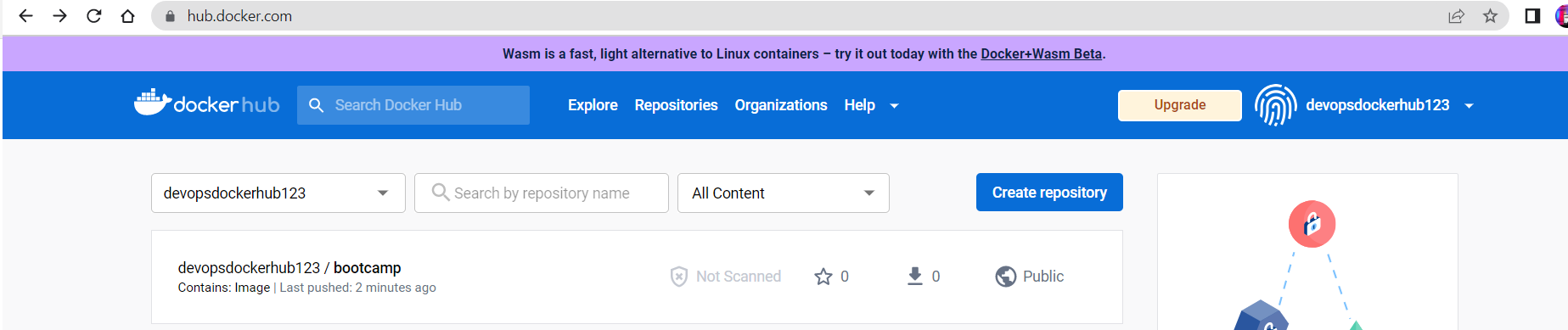
docker tag bootcamp:2022 devopsdockerhub123/bootcamp:2022

docker push





Pushed image on Docker Hub for future use



# **Containerization of Web Page & expose it**

Now create a pod and expose it .

kubectl run myweb --image devopsdockerhub123/bootcamp:2022

kubectl expose pod myweb --port=80 --type NodePort

Web Page is visible on port defined in range of 30001 ~ 32767



# **Appendix-A**

# Kubernetes Cluster Installation

Master hostname (master-node)

worker node1 (worker-node1)

worker node2 (worker-node2)

You must be a root user on all nodes to perform the following tasks:

`sudo su -`

**## STEP-1:**

`vim /etc/modules-load.d/containerd.conf`

Insert following in file:

overlay

br\_netfilter

save file and run these command:

`modprobe overlay`

`modprobe br\_netfilter`

**## STEP-2:**

`vim /etc/sysctl.d/kubernetes.conf`

Insert following in file:

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

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

net.ipv4.ip\_forward = 1

save file and run this command:

`sysctl --system`

`sysctl -p`

**## STEP-3**

Run following commands to enable docker repository:

`curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`

`add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"`

Install containerd in all nodes `apt update`

```shell

apt install -y containerd.io

rm -f /etc/containerd/config.toml

systemctl daemon-reload

systemctl start containerd

systemctl enable containerd

systemctl status containerd

```

**## STEP-4**

Add kubernetes repository in all nodes

`apt-get update && apt-get install -y apt-transport-https ca-certificates curl`

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

`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`

**## STEP-5**

Disable swap:

`vim /etc/fstab` (Remove swap mount point from this file)

`swapoff -a`

**## STEP-6**

Export the environment variable:

`export KUBE\_VERSION=1.23.0`

Install kubernetes `apt-get update`

```shell

apt-get install -y kubelet=${KUBE\_VERSION}-00 kubeadm=${KUBE\_VERSION}-00 kubectl=${KUBE\_VERSION}-00 kubernetes-cni=0.8.7-00

apt-mark hold kubelet kubeadm kubectl (Secure the packages for accidental removal)

systemctl enable kubelet

systemctl start kubelet

```

**## STEP-7 - Master**

`kubeadm init --kubernetes-version=${KUBE\_VERSION}` (Only on master node)

`kubeadm token create --print-join-command` (To regenrate the tokens)

**## STEP-8 - Master**

```shell

cp /etc/kubernetes/admin.conf /root/

chown $(id -u):$(id -g) /root/admin.conf

KUBECONFIG=/root/admin.conf

echo 'export KUBECONFIG=/root/admin.conf' >> /root/.bashrc

```

**## STEP-9 - Master**

`kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"`

OR

`https://github.com/weaveworks/weave/releases`

Download the daemonset yaml file of required version like following link.

`wget https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml`

then

`kubectl apply -f weave-daemonset-k8s.yaml`

# **Appendix-B**

## YAML for Multi-Tier Application

apiVersion: v1

kind: Service

metadata:

  name: db

  labels:

    app: words-db

spec:

  ports:

    - port: 5432

      targetPort: 5432

      name: db

  selector:

    app: words-db

  clusterIP: None

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: db

  labels:

    app: words-db

spec:

  selector:

    matchLabels:

      app: words-db

  template:

    metadata:

      labels:

        app: words-db

    spec:

      containers:

      - name: db

        image: dockersamples/k8s-wordsmith-db

        ports:

        - containerPort: 5432

          name: db

---

apiVersion: v1

kind: Service

metadata:

  name: words

  labels:

    app: words-api

spec:

  ports:

    - port: 8080

      targetPort: 8080

      name: api

  selector:

    app: words-api

  clusterIP: None

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: words

  labels:

    app: words-api

spec:

  replicas: 5

  selector:

    matchLabels:

      app: words-api

  template:

    metadata:

      labels:

        app: words-api

    spec:

      containers:

      - name: words

        image: dockersamples/k8s-wordsmith-api

        ports:

        - containerPort: 8080

          name: api

---

apiVersion: v1

kind: Service

metadata:

  name: web

  labels:

    app: words-web

spec:

  ports:

    - port: 8081

      targetPort: 80

      name: web

  selector:

    app: words-web

  type: LoadBalancer

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: web

  labels:

    app: words-web

spec:

  selector:

    matchLabels:

      app: words-web

  template:

    metadata:

      labels:

        app: words-web

    spec:

      containers:

      - name: web

        image: dockersamples/k8s-wordsmith-web

        ports:

        - containerPort: 80

          name: words-web