**K8s HighAvailablity**

**- with metalLB and NFS persistent volume**

**www.zippyops.com**

172-172, 5th floor Old Mahabalipuram Road

(Above Axis Bank-PTC Bus Stop)  
Thuraipakkam  
Chennai 600097

🖂[zippyops@gmail.com](mailto:zippyops@gmail.com)

✆+91 7010585768

Kubeadm is a tool which is part of the Kubernetes project. It is designed to help with the deployment of Kubernetes. It is currently a work in progress and it has some limitations. One of these limitations is that it doesn't support multi-master (high availability) configuration. This tutorial will go through the steps allowing to work around this limitation.

In this Lab we are going to Install and configure a multi-master Kubernetes cluster with kubeadm.

# Prerequisites

For this lab, we will use a standard Ubuntu 16.04 installation as a base image for the seven machines needed. The machines will all be configured on the same network, this network needs to have access to the Internet.

The first machine needed is the machine on which the HAProxy load balancer will be installed. We will assign the IP 192.168.1.93 to this machine.

We also need three Kubernetes master nodes. These machines will have the IPs192.168.1.90, 192.168.1.91, and 192.168.1.92.

Finally, we will also have three Kubernetes worker nodes with the IPs 192.168.1.94, 192.168.1.95, and 192.168.1.96.

For this lab we are going to use three master (minimum three is need) embedded etcd and three worker nodes that will work fine.

Use unique machine to generate all the necessary certificates to manage the Kubernetes cluster. If you don't have machine you can use the HAProxy machine to do the same thing.

## System requirments

* 3-Master => 2 CPU , 2GB RAM
* 3-Nodes => 1 CPU, 1GB RAM
* 1-Load balancer (node) => 1 CPU, 1GB RAM
* 1- NFS (node) => 1 CPU, 1 GB RAM

# Installing HAProxy load balancer

## Installing the client tools

We will need two tools on the client machine: the Cloud Flare SSL tool to generate the different certificates, and the Kubernetes client, kubectl, to manage the Kubernetes cluster.

## Installing cfssl

Download the binaries.

$ wget https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64

$ wget https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64

Add the execution permission to the binaries.

$ chmod +x cfssl\*

Move the binaries to /usr/local/bin.

sudo mv cfssl\_linux-amd64 /usr/local/bin/cfssl

sudo mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson

Verify the installation.

$ cfssl version

*Version: 1.2.0*

*Revision: dev*

*Runtime: go1.6*

## Installing kubectl

Download the binary.

$ wget <https://storage.googleapis.com/kubernetes->release/release/v1.12.1/bin/linux/amd64/kubectl

 Add the execution permission to the binary.

chmod +x kubectl

Move the binary to /usr/local/bin.

$ sudo mv kubectl /usr/local/bin

Verify the installation.

$ kubectl version

Client Version: v1.13.4

## Install load balancer

As we will deploy three Kubernetes master nodes, we need to deploy an HAPRoxy load balancer in front of them to distribute the traffic.

Folloe these steps to all Master nodes.

Update the machine.

$ sudo apt-get update

$ sudo apt-get upgrade

Install HAProxy.

$ sudo apt-get install haproxy

Configure HAProxy to load balance the traffic between the three Kubernetes master nodes.

$ sudo vim /etc/haproxy/haproxy.cfg

global

...

default

...

frontend kubernetes

bind 192.168.1.93:6443

option tcplog

mode tcp

default\_backend kubernetes-master-nodes

backend kubernetes-master-nodes

mode tcp

balance roundrobin

option tcp-check

server kubmaster1.zippyops.com 192.168.1.90:6443 check fall 3 rise 2

server kubmaster2.zippyops.com 192.168.1.91:6443 check fall 3 rise 2

server kubmaster3.zippyops.com 192.168.1.92:6443 check fall 3 rise 2

Restart HAProxy.

sudo systemctl restart haproxy

## Generating the TLS certificates

These steps can be done on your Loadbalancer if you have one or on the HAProxy machine depending on where you installed the cfssl tool.

## Creating a certificate authority

Create the certificate authority configuration file.

vim ca-config.json

{

"signing": {

"default": {

"expiry": "8760h"

},

"profiles": {

"kubernetes": {

"usages": ["signing", "key encipherment", "server auth", "client auth"],

"expiry": "8760h"

}

}

}

}

Create the certificate authority signing request configuration file.

vim ca-csr.json

{

"CN": "Kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "IE",

"L": "Cork",

"O": "Kubernetes",

"OU": "CA",

"ST": "Cork Co."

}

]

}

Generate the certificate authority certificate and private key.

cfssl gencert -initca ca-csr.json | cfssljson -bare ca

Verify that the ca-key.pem and the ca.pem were generated.

ls –la

## Creating the certificate for the Etcd cluster

Create the certificate signing request configuration file.

vim kubernetes-csr.json

{

"CN": "kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "IE",

"L": "Cork",

"O": "Kubernetes",

"OU": "Kubernetes",

"ST": "Cork Co."

}

]

}

Generate the certificate and private key.

$ cfssl gencert \

-ca=ca.pem \

-ca-key=ca-key.pem \

-config=ca-config.json \

-hostname=192.168.1.90,192.168.1.91,192.168.1.92,192.168.1.93,127.0.0.1,kubernetes.default \

-profile=kubernetes kubernetes-csr.json | \

cfssljson -bare kubernetes

Verify that the kubernetes-key.pem and the kubernetes.pem file were generated.

ls –la

Copy the certificate to each nodes.

$ scp ca.pem kubernetes.pem kubernetes-key.pem zippyops@192.168.1.90:/home

$ scp ca.pem kubernetes.pem kubernetes-key.pem zippyops@192.168.1.91:/home

$ scp ca.pem kubernetes.pem kubernetes-key.pem zippyops@192.168.1.92:/home

$ scp ca.pem kubernetes.pem kubernetes-key.pem zippyops@192.168.1.94:/home

$ scp ca.pem kubernetes.pem kubernetes-key.pem zippyops@192.168.1.95:/home

$ scp ca.pem kubernetes.pem kubernetes-key.pem zippyops@192.168.1.96:/home

# installing the kubeadm for all masters

## Installing Docker

Add the Docker repository key and repo

$curl -fsSL https://[download.docker.com/linux/ubuntu/gpg](http://download.docker.com/linux/ubuntu/gpg) | apt-key add -

$ add-apt-repository \

"deb <https://download.docker.com/linux/>$(. /etc/os-release; echo "$ID") \

$(lsb\_release -cs) \

stable"

Update the list of packages.

$ apt-get update

Install Docker 17.03.

$ apt-get install -y docker-ce=$(apt-cache madison docker-ce | grep 17.03 | head -1 | awk '{print $3}')

## Installing kubeadm, kublet, and kubectl

 Add the Google repository key.

$ curl -s https://[packages.cloud.google.com/apt/doc/apt-key.gpg](http://packages.cloud.google.com/apt/doc/apt-key.gpg) | apt-key add -

Add the Google repository.

$ vim /etc/apt/sources.list.d/kubernetes.list

Add this line to repo kubernetes.list

deb [http://apt.kubernetes.io](http://apt.kubernetes.io/) kubernetes-xenial main

Update the list of packages. and Install kubelet, kubeadm and kubectl.

$ apt-get update && apt-get install kubelet kubeadm kubectl –y

Disable the swap.

$ swapoff –a

$ sed -i '/ swap / s/^/#/' /etc/fstab

Follow these above steps to **other two master nodes**.

# Installing and configuring Etcd on all masters

## Installing and configuring Etcd on the 192.168.1.90 machine

Create a configuration directory for Etcd.

$ mkdir /etc/etcd /var/lib/etcd

Move the certificates to the configuration directory.

$ mv /home/ca.pem /home/kubernetes.pem /home/kubernetes-key.pem /etc/etcd

Download the etcd binaries.

$ wget https://[github.com/coreos](http://github.com/coreos)/etcd/releases/download/v3.3.9/etcd-v3.3.9-linux-amd64.tar.gz

Extract the etcd archive.

$ tar xvzf etcd-v3.3.9-linux-amd64.tar.gz

Move the etcd binaries to /usr/local/bin.

$ mv etcd-v3.3.9-linux-amd64/etcd\* /usr/local/bin/

Create an etcd systemd unit file.

$ vim /etc/systemd/system/etcd.service

[Unit]

Description=etcd

Documentation=https://[github.com/coreos](http://github.com/coreos)

[Service]

ExecStart=/usr/local/bin/etcd \

--name 192.168.1.90 \

--cert-file=/etc/etcd/kubernetes.pem \

--key-file=/etc/etcd/kubernetes-key.pem \

--peer-cert-file=/etc/etcd/kubernetes.pem \

--peer-key-file=/etc/etcd/kubernetes-key.pem \

--trusted-ca-file=/etc/etcd/ca.pem \

--peer-trusted-ca-file=/etc/etcd/ca.pem \

--peer-client-cert-auth \

--client-cert-auth \

--initial-advertise-peer-urls https://192.168.1.90:2380 \

--listen-peer-urls https://192.168.1.90:2380 \

--listen-client-urls https://192.168.1.90:2379,http://127.0.0.1:2379 \

--advertise-client-urls https://192.168.1.90:2379 \

--initial-cluster-token etcd-cluster-0 \

--initial-cluster 192.168.1.90=https://192.168.1.90:2380,192.168.1.91=https://192.168.1.91:2380,192.168.1.92=https://192.168.1.92:2380 \

--initial-cluster-state new \

--data-dir=/var/lib/etcd

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

Reload the daemon configuration.

$ systemctl daemon-reload

Enable etcd to start at boot time and Start etcd.

$ systemctl enable etcd && systemctl start etcd

## Installing and configuring Etcd on the 192.168.1.91 machine

Create a configuration directory for Etcd.

$ mkdir /etc/etcd /var/lib/etcd

Move the certificates to the configuration directory.

$ mv /home/ca.pem /home/kubernetes.pem /home/kubernetes-key.pem /etc/etcd

Download the etcd binaries.

$ wget https://[github.com/coreos](http://github.com/coreos)/etcd/releases/download/v3.3.9/etcd-v3.3.9-linux-amd64.tar.gz

Extract the etcd archive.

$ tar xvzf etcd-v3.3.9-linux-amd64.tar.gz

Move the etcd binaries to /usr/local/bin.

$ mv etcd-v3.3.9-linux-amd64/etcd\* /usr/local/bin/

Create an etcd systemd unit file.

$ vim /etc/systemd/system/etcd.service

[Unit]

Description=etcd

Documentation=https://[github.com/coreos](http://github.com/coreos)

[Service]

ExecStart=/usr/local/bin/etcd \

--name 192.168.1.91 \

--cert-file=/etc/etcd/kubernetes.pem \

--key-file=/etc/etcd/kubernetes-key.pem \

--peer-cert-file=/etc/etcd/kubernetes.pem \

--peer-key-file=/etc/etcd/kubernetes-key.pem \

--trusted-ca-file=/etc/etcd/ca.pem \

--peer-trusted-ca-file=/etc/etcd/ca.pem \

--peer-client-cert-auth \

--client-cert-auth \

--initial-advertise-peer-urls https://192.168.1.91:2380 \

--listen-peer-urls https://192.168.1.91:2380 \

--listen-client-urls https://192.168.1.91:2379,http://127.0.0.1:2379 \

--advertise-client-urls https://192.168.1.91:2379 \

--initial-cluster-token etcd-cluster-0 \

--initial-cluster 192.168.1.90=https://192.168.1.90:2380,192.168.1.91=https://192.168.1.91:2380,192.168.1.92=https://192.168.1.92:2380 \

--initial-cluster-state new \

--data-dir=/var/lib/etcd

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

Reload the daemon configuration.

$ systemctl daemon-reload

Enable etcd to start at boot time and Start etcd.

$ systemctl enable etcd && systemctl start etcd

## Installing and configuring Etcd on the 192.168.1.92 machine

Create a configuration directory for Etcd.

$ mkdir /etc/etcd /var/lib/etcd

Move the certificates to the configuration directory.

$ mv /home/ca.pem /home/kubernetes.pem /home/kubernetes-key.pem /etc/etcd

Download the etcd binaries.

$ wget https://[github.com/coreos](http://github.com/coreos)/etcd/releases/download/v3.3.9/etcd-v3.3.9-linux-amd64.tar.gz

Extract the etcd archive.

$ tar xvzf etcd-v3.3.9-linux-amd64.tar.gz

Move the etcd binaries to /usr/local/bin.

$ mv etcd-v3.3.9-linux-amd64/etcd\* /usr/local/bin/

Create an etcd systemd unit file.

$ **vim /etc/systemd/system/etcd.service**

[Unit]

Description=etcd

Documentation=https://[github.com/coreos](http://github.com/coreos)

[Service]

ExecStart=/usr/local/bin/etcd \

--name 192.168.1.92 \

--cert-file=/etc/etcd/kubernetes.pem \

--key-file=/etc/etcd/kubernetes-key.pem \

--peer-cert-file=/etc/etcd/kubernetes.pem \

--peer-key-file=/etc/etcd/kubernetes-key.pem \

--trusted-ca-file=/etc/etcd/ca.pem \

--peer-trusted-ca-file=/etc/etcd/ca.pem \

--peer-client-cert-auth \

--client-cert-auth \

--initial-advertise-peer-urls https://192.168.1.92:2380 \

--listen-peer-urls https://192.168.1.92:2380 \

--listen-client-urls https://192.168.1.92:2379,http://127.0.0.1:2379 \

--advertise-client-urls https://192.168.1.92:2379 \

--initial-cluster-token etcd-cluster-0 \

--initial-cluster 192.168.1.90=https://192.168.1.90:2380,192.168.1.91=https://192.168.1.91:2380,192.168.1.92=https://192.168.1.92:2380 \

--initial-cluster-state new \

--data-dir=/var/lib/etcd

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

Reload the daemon configuration.

$ systemctl daemon-reload

Enable etcd to start at boot time and Start etcd.

$ systemctl enable etcd && systemctl start etcd

Verify that the cluster is up and running.

$ ETCDCTL\_API=3 etcdctl member list

*31ed2fadd07c4469, started, 192.168.1.90, https://192.168.1.90:2380, https://192.168.1.90:2379*

*608fdbe685b1ab6e, started, 192.168.1.91, https://192.168.1.91:2380, https://192.168.1.91:2379*

*d71352a6aad35c57, started, 192.168.1.92, https://192.168.1.92:2380, https://192.168.1.92:2379*

# Initializing the all 3 master nodes

## initializing the 192.168.1.90 master node

Create the configuration file for kubeadm.

$ vim config.yaml

apiVersion: [kubeadm.k8s.io/v1alpha3](http://kubeadm.k8s.io/v1alpha3)

kind: ClusterConfiguration

kubernetesVersion: stable

apiServerCertSANs:

- 192.168.1.93

controlPlaneEndpoint: "192.168.1.93:6443"

etcd:

external:

endpoints:

- https://192.168.1.90:2379

- https://192.168.1.91:2379

- https://192.168.1.92:2379

caFile: /etc/etcd/ca.pem

certFile: /etc/etcd/kubernetes.pem

keyFile: /etc/etcd/kubernetes-key.pem

networking:

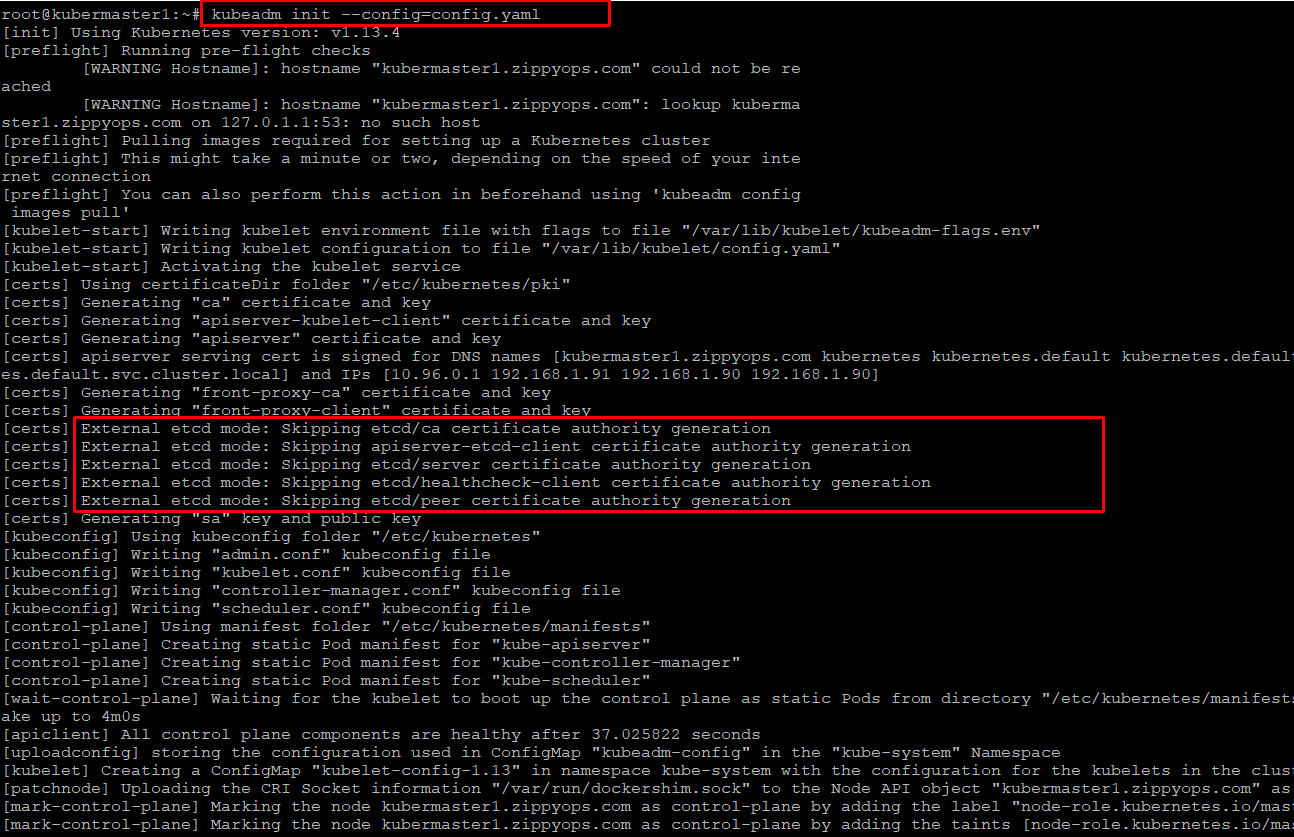
podSubnet: 10.30.0.0/24

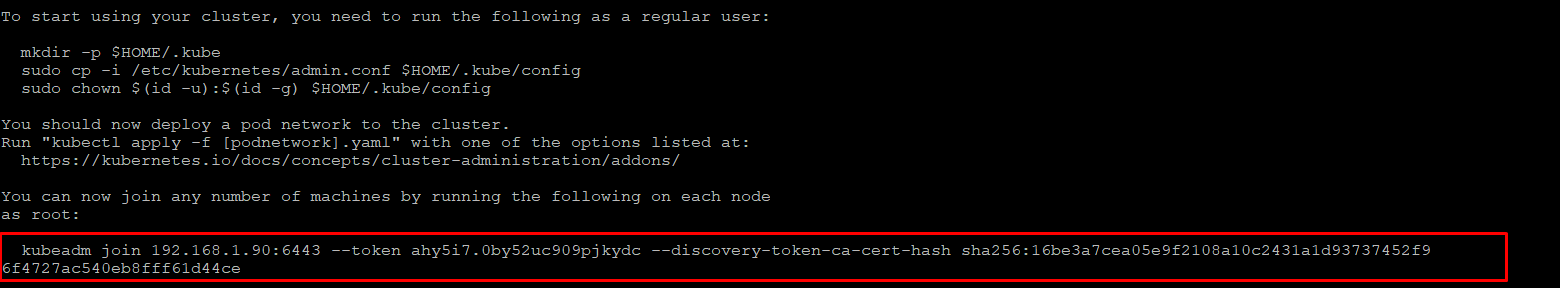
apiServerExtraArgs:

apiserver-count: "3"

Initialize the machine as a master node

kubeadm init --config=config.yaml





Copy the certificates to the two other masters.

$ scp -r /etc/kubernetes/pki zippyops@192.168.1.91:/home

$ scp -r /etc/kubernetes/pki [zippyops@192.168.1.92:/home](mailto:zippyops@192.168.1.92:/home)

## initializing the 192.168.1.91 master node

Remove the apiserver.crt and apiserver.key.

$ rm /home/pki/apiserver.\*

Move the certificates to the /etc/kubernetes directory.

$ mv /home/pki /etc/kubernetes/

Create the configuration file for kubeadm.

$ vim config.yaml

apiVersion: [kubeadm.k8s.io/v1alpha3](http://kubeadm.k8s.io/v1alpha3)

kind: ClusterConfiguration

kubernetesVersion: stable

apiServerCertSANs:

- 192.168.1.93

controlPlaneEndpoint: "192.168.1.93:6443"

etcd:

external:

endpoints:

- https://192.168.1.90:2379

- https://192.168.1.91:2379

- https://192.168.1.92:2379

caFile: /etc/etcd/ca.pem

certFile: /etc/etcd/kubernetes.pem

keyFile: /etc/etcd/kubernetes-key.pem

networking:

podSubnet: 10.30.0.0/24

apiServerExtraArgs:

apiserver-count: "3"

Initialize the machine as a master node.

$ kubeadm init --config=config.yaml

## Initializing the 192.168.1.92 master node

Remove the apiserver.crt and apiserver.key.

$ rm /home/pki/apiserver.\*

Move the certificates to the /etc/kubernetes directory.

$ mv /home/pki /etc/kubernetes/

Create the configuration file for kubeadm.

$ vim config.yaml

apiVersion: [kubeadm.k8s.io/v1alpha3](http://kubeadm.k8s.io/v1alpha3)

kind: ClusterConfiguration

kubernetesVersion: stable

apiServerCertSANs:

- 192.168.1.93

controlPlaneEndpoint: "192.168.1.93:6443"

etcd:

external:

endpoints:

- https://192.168.1.90:2379

- https://192.168.1.91:2379

- https://192.168.1.92:2379

caFile: /etc/etcd/ca.pem

certFile: /etc/etcd/kubernetes.pem

keyFile: /etc/etcd/kubernetes-key.pem

networking:

podSubnet: 10.30.0.0/24

apiServerExtraArgs:

apiserver-count: "3"

Initialize the machine as a master node.

$ kubeadm init --config=config.yaml

Copy the "kubeadm join" command line printed as the result of the previous command.

# installing the kubeadm for all kubernetes NODES

## Installing Docker

Add the Docker repository key and repo

$ curl -fsSL https://[download.docker.com/linux/ubuntu/gpg](http://download.docker.com/linux/ubuntu/gpg) | apt-key add -

$ add-apt-repository \

"deb <https://download.docker.com/linux/>$(. /etc/os-release; echo "$ID") \

$(lsb\_release -cs) \

stable"

Update the list of packages.

$ apt-get update

Install Docker 17.03.

$ apt-get install -y docker-ce=$(apt-cache madison docker-ce | grep 17.03 | head -1 | awk '{print $3}')

## Installing kubeadm, kublet, and kubectl

 Add the Google repository key.

$ curl -s https://[packages.cloud.google.com/apt/doc/apt-key.gpg](http://packages.cloud.google.com/apt/doc/apt-key.gpg) | apt-key add -

Add the Google repository.

$ vim /etc/apt/sources.list.d/kubernetes.list

deb [http://apt.kubernetes.io](http://apt.kubernetes.io/) kubernetes-xenial main

Update the list of packages. and Install kubelet, kubeadm and kubectl.

$ apt-get update && apt-get install kubelet kubeadm kubectl –y

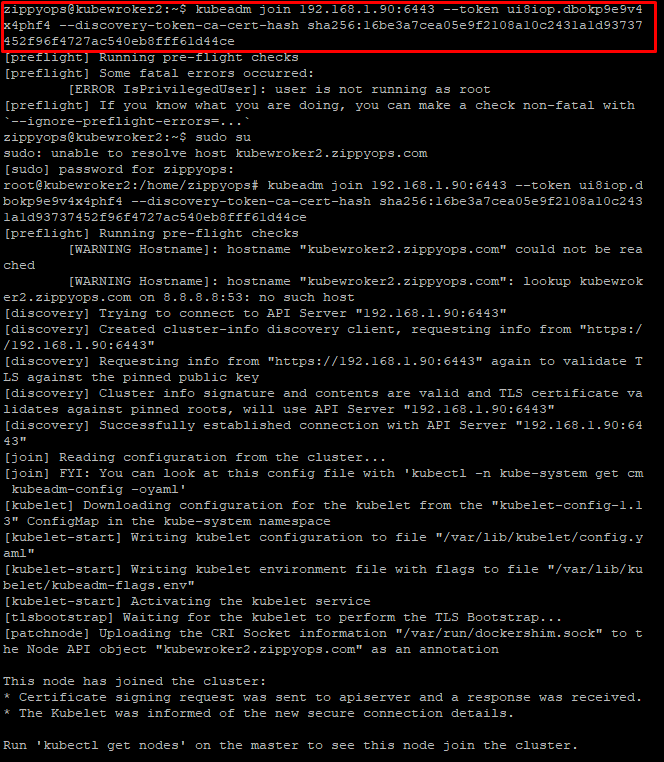
Disable the swap.

$ swapoff –a

$ sed -i '/ swap / s/^/#/' /etc/fstab

Execute the "kubeadm join" command that you copied from the last step of the initialization of the masters. (discovery token must be same on all master) cert generate for load balancer ip address

$ kubeadm join 192.168.1.93:6443 --token [your\_token] --discovery-token-ca-cert-hash sha256:[your\_token\_ca\_cert\_hash]



# Verifying that the workers joined the cluster on master

 To one of the master node.

$ kubectl --kubeconfig /etc/kubernetes/admin.conf get nodes

*NAME STATUS ROLES AGE VERSION*

*k8s-kubeadm-master-0 NotReady master 1h v1.12.1*

*k8s-kubeadm-master-1 NotReady master 1h v1.12.1*

*k8s-kubeadm-master-2 NotReady master 1h v1.12.1*

*k8s-kubeadm-worker-0 NotReady 2m v1.12.1*

*k8s-kubeadm-worker-1 NotReady 1m v1.12.1*

*k8s-kubeadm-worker-2 NotReady 1m v1.12.1*

The status of the nodes is NotReady as we haven't configured the networking overlay yet.

Add permissions to the admin.conf file.

$ chmod +r /etc/kubernetes/admin.conf

 From the client machine, copy the configuration file

$ scp root@192.168.1.90:/etc/kubernetes/admin.conf .

Create the kubectl configuration directory.

$ mkdir ~/.kube

Move the configuration file to the configuration directory.

$ mv admin.conf ~/.kube/config

Modify the permissions of the configuration file

$ chmod 600 ~/.kube/config

Go back to the SSH session on the master and change back the permissions of the configuration file

$ sudo chmod 600 /etc/kubernetes/admin.conf

check that you can access the Kubernetes API from the client machine.

$ kubectl get nodes

*NAME STATUS ROLES AGE VERSION*

*k8s-kubeadm-master-0 NotReady master 1h v1.12.1*

*k8s-kubeadm-master-1 NotReady master 1h v1.12.1*

*k8s-kubeadm-master-2 NotReady master 1h v1.12.1*

*k8s-kubeadm-worker-0 NotReady 2m v1.12.1*

*k8s-kubeadm-worker-1 NotReady 1m v1.12.1*

*k8s-kubeadm-worker-2 NotReady 1m v1.12.1*

# Deploying the overlay network

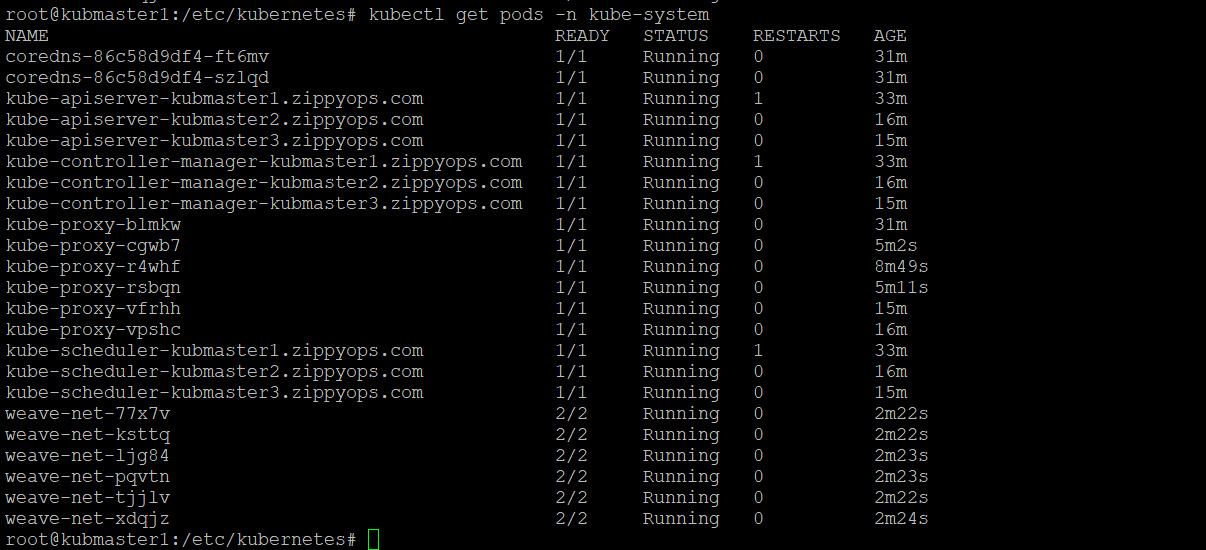
We are going to use Weavenet as the overlay network. You can also use static route or another overlay network tool like Calico or Flannel.

Deploy the overlay network pods from the client(master) machine

$ kubectl apply -f https://[git.io/weave](http://git.io/weave)-kube-1.6

Check that the pods are deployed properly

$ kubectl get pods -n kube-system



Check that the nodes are in Ready state.

$ kubectl get nodes

*NAME STATUS ROLES AGE VERSION*

*kubmaster1.zippyops.com Ready master 2d20h v1.13.4*

*kubmaster2.zippyops.com Ready master 2d19h v1.13.4*

*kubmaster3.zippyops.com Ready master 2d19h v1.13.4*

*kubwork1.zippyops.com Ready <none> 2d19h v1.13.4*

*kubwork2.zippyops.com Ready <none> 2d19h v1.13.4*

*kubwork3.zippyops.com Ready <none> 2d19h v1.13.4*

# Installing Kubernetes add-ons

We will deploy two Kubernetes add-ons on our new cluster: the dashboard add-on to have a graphical view of the cluster, and the Heapster add-on to monitor our workload

## Installing the Kubernetes dashboard

Create the Kubernetes dashboard manifest.

$ vim kubernetes-dashboard.yaml

# Copyright 2017 The Kubernetes Authors.

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

# <http://www.apache.org/licenses/LICENSE->2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

# Configuration to deploy release version of the Dashboard UI compatible with

# Kubernetes 1.8.

#

# Example usage: kubectl create -f

# ------------------- Dashboard Secret ------------------- #

apiVersion: v1

kind: Secret

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard-certs

namespace: kube-system

type: Opaque

---

# ------------------- Dashboard Service Account ------------------- #

apiVersion: v1

kind: ServiceAccount

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

---

# ------------------- Dashboard Role & Role Binding ------------------- #

kind: Role

apiVersion: rbac.authorization.[k8s.io/v1](http://k8s.io/v1)

metadata:

name: kubernetes-dashboard-minimal

namespace: kube-system

rules:

# Allow Dashboard to create 'kubernetes-dashboard-key-holder' secret.

- apiGroups: [""]

resources: ["secrets"]

verbs: ["create"]

# Allow Dashboard to create 'kubernetes-dashboard-settings' config map.

- apiGroups: [""]

resources: ["configmaps"]

verbs: ["create"]

# Allow Dashboard to get, update and delete Dashboard exclusive secrets.

- apiGroups: [""]

resources: ["secrets"]

resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-dashboard-certs"]

verbs: ["get", "update", "delete"]

# Allow Dashboard to get and update 'kubernetes-dashboard-settings' config map.

- apiGroups: [""]

resources: ["configmaps"]

resourceNames: ["kubernetes-dashboard-settings"]

verbs: ["get", "update"]

# Allow Dashboard to get metrics from heapster.

- apiGroups: [""]

resources: ["services"]

resourceNames: ["heapster"]

verbs: ["proxy"]

- apiGroups: [""]

resources: ["services/proxy"]

resourceNames: ["heapster", "http:heapster:", "https:heapster:"]

verbs: ["get"]

---

apiVersion: rbac.authorization.[k8s.io/v1](http://k8s.io/v1)

kind: RoleBinding

metadata:

name: kubernetes-dashboard-minimal

namespace: kube-system

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: kubernetes-dashboard-minimal

subjects:

- kind: ServiceAccount

name: kubernetes-dashboard

namespace: kube-system

---

# ------------------- Dashboard Deployment ------------------- #

kind: Deployment

apiVersion: apps/v1beta2

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

spec:

replicas: 1

revisionHistoryLimit: 10

selector:

matchLabels:

k8s-app: kubernetes-dashboard

template:

metadata:

labels:

k8s-app: kubernetes-dashboard

spec:

containers:

- name: kubernetes-dashboard

image: [k8s.gcr.io/kubernetes-dashboard-amd64:v1](http://k8s.gcr.io/kubernetes-dashboard-amd64:v1).8.3

ports:

- containerPort: 8443

protocol: TCP

args:

- --auto-generate-certificates

# Uncomment the following line to manually specify Kubernetes API server Host

# If not specified, Dashboard will attempt to auto discover the API server and connect

# to it. Uncomment only if the default does not work.

# - --apiserver-host=[http://my-address:port](http://my-address:port/)

volumeMounts:

- name: kubernetes-dashboard-certs

mountPath: /certs

# Create on-disk volume to store exec logs

- mountPath: /tmp

name: tmp-volume

livenessProbe:

httpGet:

scheme: HTTPS

path: /

port: 8443

initialDelaySeconds: 30

timeoutSeconds: 30

volumes:

- name: kubernetes-dashboard-certs

secret:

secretName: kubernetes-dashboard-certs

- name: tmp-volume

emptyDir: {}

serviceAccountName: kubernetes-dashboard

# Comment the following tolerations if Dashboard must not be deployed on master

tolerations:

- key: node-role.kubernetes.io/master

effect: NoSchedule

---

# ------------------- Dashboard Service ------------------- #

kind: Service

apiVersion: v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

spec:

ports:

- port: 443

targetPort: 8443

selector:

k8s-app: kubernetes-dashboard

Deploy the dashboard.

$ kubectl create -f kubernetes-dashboard.yaml

## Installing Heapster

Create a manifest for Heapster

$ vim heapster.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: heapster

namespace: kube-system

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: heapster

namespace: kube-system

spec:

replicas: 1

template:

metadata:

labels:

task: monitoring

k8s-app: heapster

spec:

serviceAccountName: heapster

containers:

- name: heapster

image: [gcr.io/google\_containers/heapster-amd64:v1.4.2](http://gcr.io/google_containers/heapster-amd64:v1.4.2)

imagePullPolicy: IfNotPresent

command:

- /heapster

- --source=kubernetes.summary\_api:''?useServiceAccount=true&kubeletHttps=true&kubeletPort=10250&insecure=true

---

apiVersion: v1

kind: Service

metadata:

labels:

task: monitoring

# For use as a Cluster add-on (<https://github.com/kubernetes/kubernetes/tree/>master/cluster/addons)

# If you are NOT using this as an addon, you should comment out this line.

[kubernetes.io/cluster-service](http://kubernetes.io/cluster-service): 'true'

[kubernetes.io/name](http://kubernetes.io/name): Heapster

name: heapster

namespace: kube-system

spec:

ports:

- port: 80

targetPort: 8082

selector:

k8s-app: heapster

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.[k8s.io/v1beta1](http://k8s.io/v1beta1)

metadata:

name: heapster

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: system:heapster

subjects:

- kind: ServiceAccount

name: heapster

namespace: kube-system

 Deploy Heapster.

$ kubectl create -f heapster.yaml

Edit the Heapster RBAC role and add the get permission on the nodes statistic at the end.

$ kubectl edit clusterrole system:heapster

...

- apiGroups:

- ""

resources:

- nodes/**stats**

verbs:

- **get**

## Accessing the Kubernetes dashboard

 Create an admin user manifest.

$ vim kubernetes-dashboard-admin.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

namespace: kube-system

---

apiVersion: [rbac.authorization.k8s.io/v1beta1](http://rbac.authorization.k8s.io/v1beta1)

kind: ClusterRoleBinding

metadata:

name: admin-user

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: admin-user

namespace: kube-system

Create the admin user

$ kubectl create -f kubernetes-dashboard-admin.yaml

Get the admin user token.

$ kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')

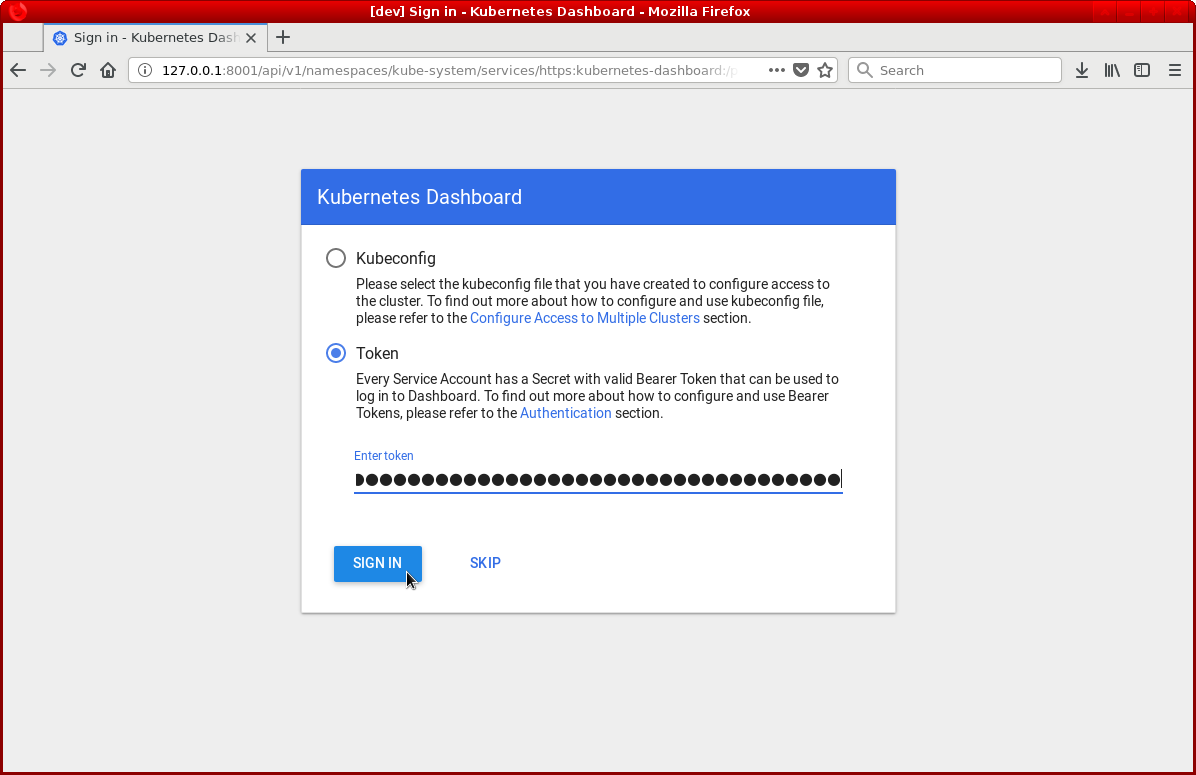
Copy the token.

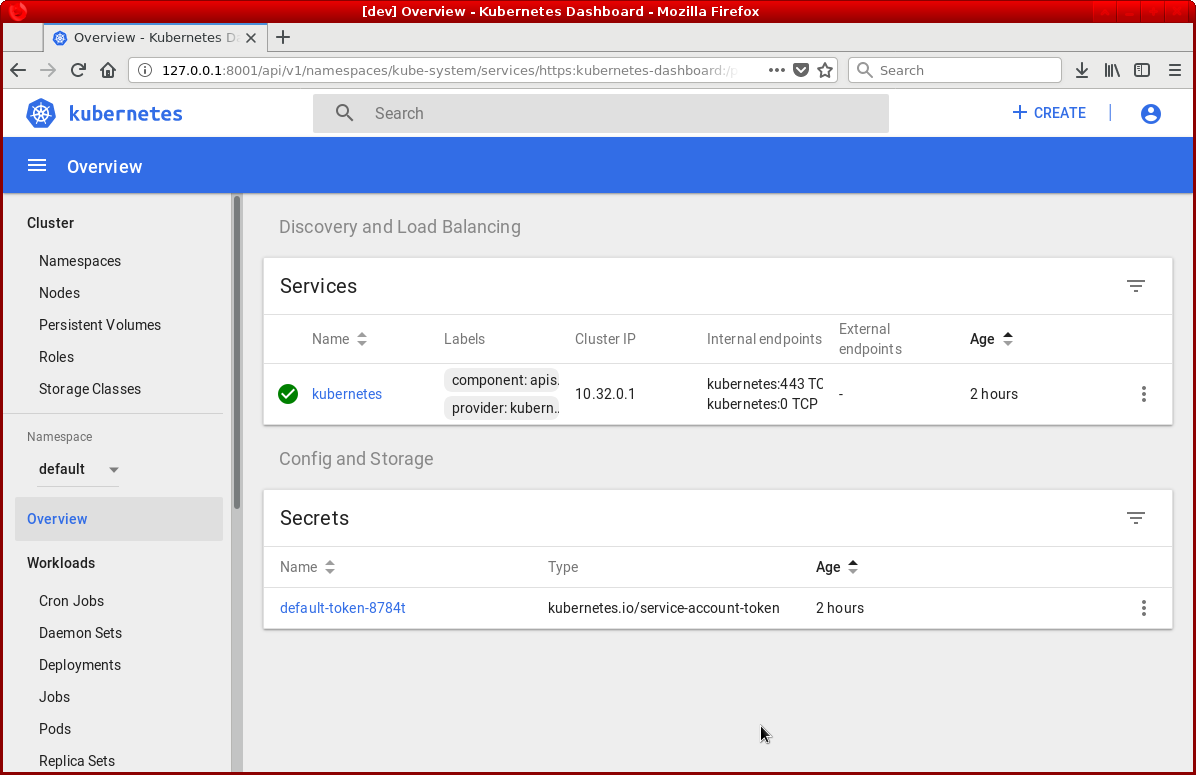
Start the proxy to access the dashboard.

$ kubectl proxy

Browse to  <http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy>.

Select Token and paste the token





# Install Kubernetes CLI on Windows 10

If you are looking to access Kubernetes Cluster from your windows machine. Look no further! I will show you how to install Kubernetes command line utilities by leveraging Chocolatey [installer](https://chocolatey.org/install).

Note. I will be using Windows 10 to demonstrate.

Now lets go ahead and get started by opening PowerShell as administrator and execute below command.

$ Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('<https://chocolatey.org/install.ps1>'))

Now that Chocolatey has been installed, we will go ahead with Kubernetes CLI setup.

## Install Kubernetes CLI

Open PowerShell as an administrator and execute below command

$ choco install kubernetes-cli

You will be prompted to confirm if you want to proceed forward with the installation. Go ahead and say yes by typing Y and hit enter.

## Connect to Kubernetes Cluster with Kubectl

Once you have install Kubernetes cli. Go to your Kubernetes master node and copy config file from ~/.kube/config to your windows machine to any location. We will move that file to required location once we create .kube directory on windows. Follow below steps.

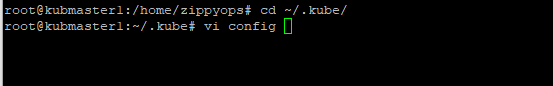
Open PowerShell as an administrator and execute below commands.

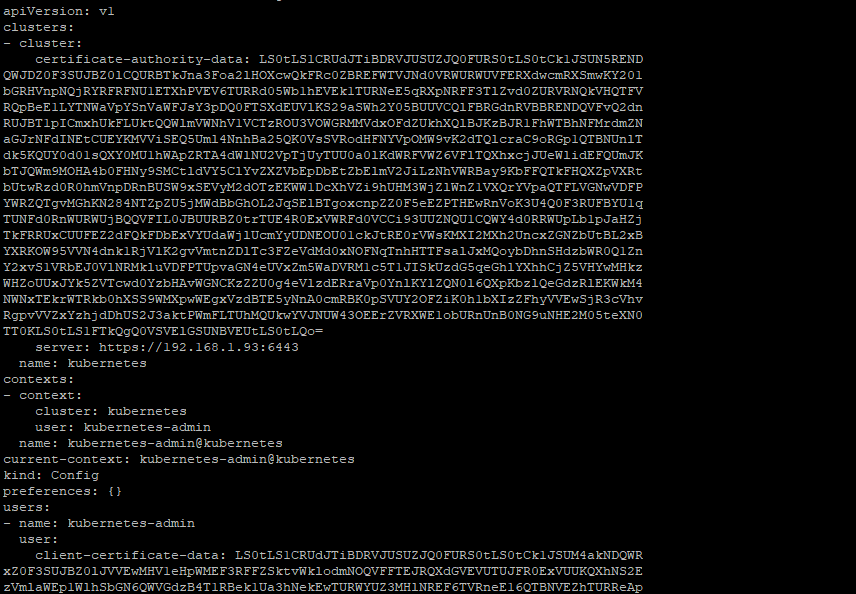
$ cd ~

Above command will take you to your user home directory. In user home directory create folder called .kube. If it already exist, you can skip this step.

$ mkdir .kube

Once the above directory has been created, we need to copy config file from Kubernetes master node to .kube folder. Earlier I mentioned to copy config file to your windows machine. Take that file and drop it on your under ~\.kube location path. In windows the config file should be ALLFILES format





The above config file should be copy and paste it to windows 10

## Basic operations

After you have followed steps as shown above, let's go ahead and test connectivity with your Kubernetes cluster.

$ kubectl.exe config get-clusters

If above command returns name of the cluster, then you have applied changes successfully. Below command will get information from your master node.

$ kubectl.exe version -o yaml

For me I received following output. Yours may vary depending on your cluster configuration.

clientVersion:

buildDate: 2018-01-04T11:52:23Z

compiler: gc

gitCommit: 3a1c9449a956b6026f075fa3134ff92f7d55f812

gitTreeState: clean

gitVersion: v1.9.1

goVersion: go1.9.2

major: "1"

minor: "9"

platform: windows/amd64

serverVersion:

buildDate: 2018-01-18T09:42:01Z

compiler: gc

gitCommit: 5fa2db2bd46ac79e5e00a4e6ed24191080aa463b

gitTreeState: clean

gitVersion: v1.9.2

goVersion: go1.9.2

major: "1"

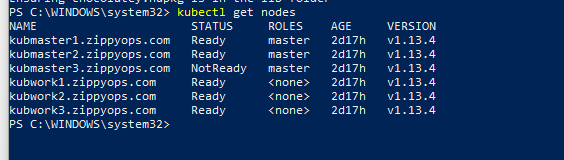
minor: "9"

platform: linux/amd64

Let's execute one more command to ensure we are successfully connected to the Kubenetes cluster.

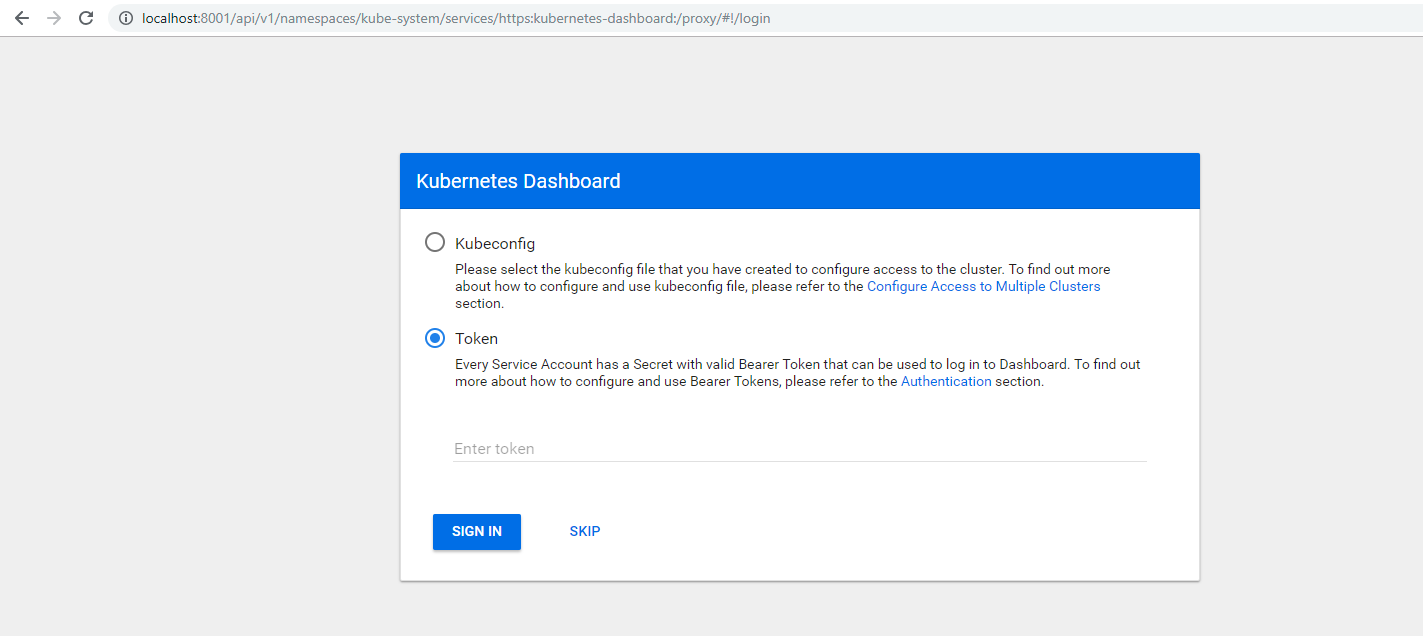
$ kubectl.exe get nodes

If you received something similar to what I have received below, then you are fully connected to the cluster and you can go ahead and manage your cluster from windows machine.





Browse to  <http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy>



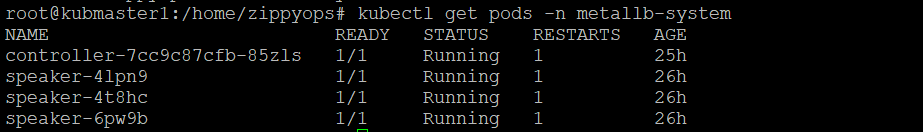
# metal load balancer on master for glowing (external ip) on premises

First we need to apply the MetalLB manifest

$ kubectl apply –f <https://raw.githubusercontent.com/google/metallb/v0.7.3/manifests/metallb.yaml>

Next we want to check that the controller and the speaker are running. we can do this by using this command

$ kubectl get pods -n metallb-system



So next we will look at our configuration on master

# cat metalconfig.yaml

apiVersion: v1

kind: ConfigMap

metadata:

namespace: metallb-system

name: config

data:

config: |

address-pools:

- name: my-ip-space

protocol: layer2

addresses:

- 192.168.1.160-192.168.1.165

Apply the config for metallb

# kubectl apply –f metalconfig.yaml

Deploy a tomcat and service for checking

# cat tomcat.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: tomcat-pod

spec:

selector:

matchLabels:

run: tomcat-pod

replicas: 3

template:

metadata:

labels:

run: tomcat-pod

spec:

containers:

- name: tomcat

image: tomcat:latest

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: tomcat-pod

labels:

run: tomcat-pod

spec:

type: LoadBalancer

ports:

- port: 8080

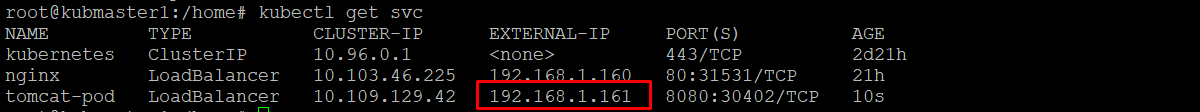
targetPort: 8080

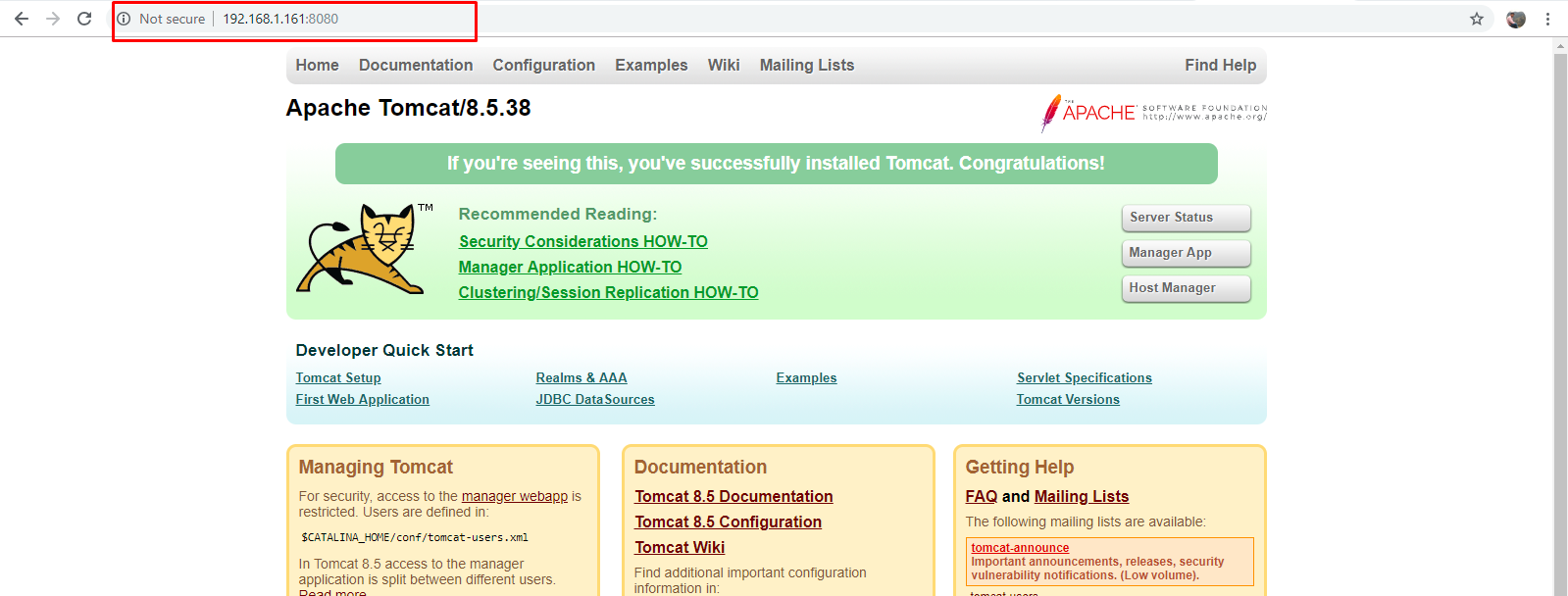
selector:

run: tomcat-pod

deploy the tomcat

# kubectl apply –f tomcat.yaml





# NFS Persistent Volume

In this Lab we are going to privision nfs server on Ubuntu 16.04 you can use your own NFS server on any platform

## Installing NFS server on Ubuntu 16.04 (new machine)

To get NFS server working you must install the server packages. run the commands below:

$ sudo apt-get update

$ sudo apt-get install nfs-kernel-server

Installing NFS client packages on the client systems.here we are going to use client as All three master nodes.

**Install NFS client on all the master nodes** using follow command,to access NFS mount points on the server,

$ sudo apt-get update

$ sudo apt-get install nfs-common

After installing the client packages, switch to the server to configure a mount point to export to the client.

Creating the folder/directory to export (share) to the NFS clients, For this Lab, we’re creating a folder called nfs/kubedata in the /srv/ directory.so run,

$ sudo mkdir –p /srv/nfs/kubedata

Since we want this location to be viewed by all clients, we’re going to remove the restrictive permissions. To do that, change the folder permission to be owned by nobody in no group.

$ sudo chown nobody:nogroup /srv/nfs/kubedata

$ sudo chmod 777 /srv/nfs/kubedata

## Configuring NFS Exports file

Now that the location is created on the host system, open NFS export file and define the client access.

Access can be granted to a single client or entire network subnet. For this Lab, we’re allowing access to the all client.

NFS export file is at /etc/exports, open the export file by running the commands below:

$ vi /etc/exports

Then add the line below:

/srv/nfs/kubedata \*(rw,sync,no\_subtree\_check,insecure)

The options in the setting above are: **rw= read/write, sync=write changes to disk before applying and no\_subtree\_check= prevents subtree checking insecure=no security, \* = all ips.**

Export the shares by running the commands below

**$ exportfs -v**

*/srv/nfs/kubedata <world>(rw,wdelay,insecure,root\_squash,no\_subtree\_check,sec=sys,rw,root\_squash,no\_all\_squash) => output*

**$ sudo exportfs –rav**

*exporting \*:/srv/nfs/kubedata => output*

Restart the NFS server by running the commands below.

$ sudo systemctl restart nfs-kernel-server

NFS server is ready to server ans share it storage… (IN master

**$ showmount -e**

*Export list for zippyops:*

*/srv/nfs/kubedata \**

## apply persistent volume(pv) on any master

Create a persistent volume for NFS

# cat pv-nfs.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv-nfs-pv1

labels:

type: nfs

spec:

storageClassName: manual

capacity:

storage: 3Gi

accessModes:

- ReadWriteMany

nfs:

server: 192.168.1.97

path: "/srv/nfs/kubedata"

to create persistent volume

# kubectl apply –f pv-nfs.yaml

To claim the persistent volume

# cat pv-claim.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: pvc-nfs-pv1

spec:

storageClassName: manual

accessModes:

- ReadWriteMany

resources:

requests:

storage: 500Mi

After claim the persistent volume create the deployment for nginx and its service with include the volume mount

# cat nginx.yaml

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

volumes:

- name: www

persistentVolumeClaim:

claimName: pvc-nfs-pv1

containers:

- name: nginx

image: nginx

volumeMounts:

- name: www

mountPath: /usr/share/nginx/html

ports:

- name: http

containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: nginx

spec:

ports:

- name: http

port: 80

protocol: TCP

targetPort: 80

selector:

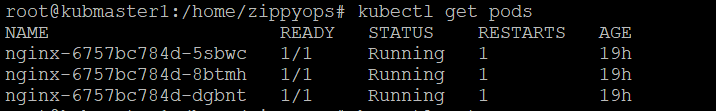
app: nginx

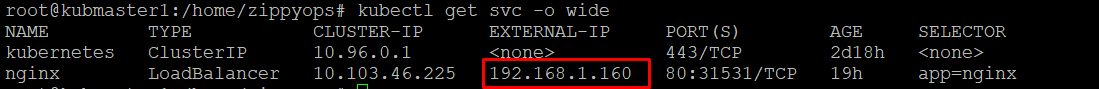
type: LoadBalancer

to run pods with volume mount

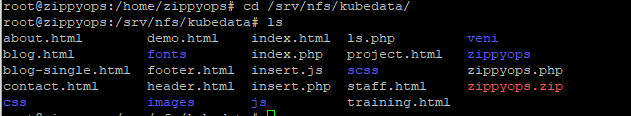
# kubectl apply –f nginx.yaml

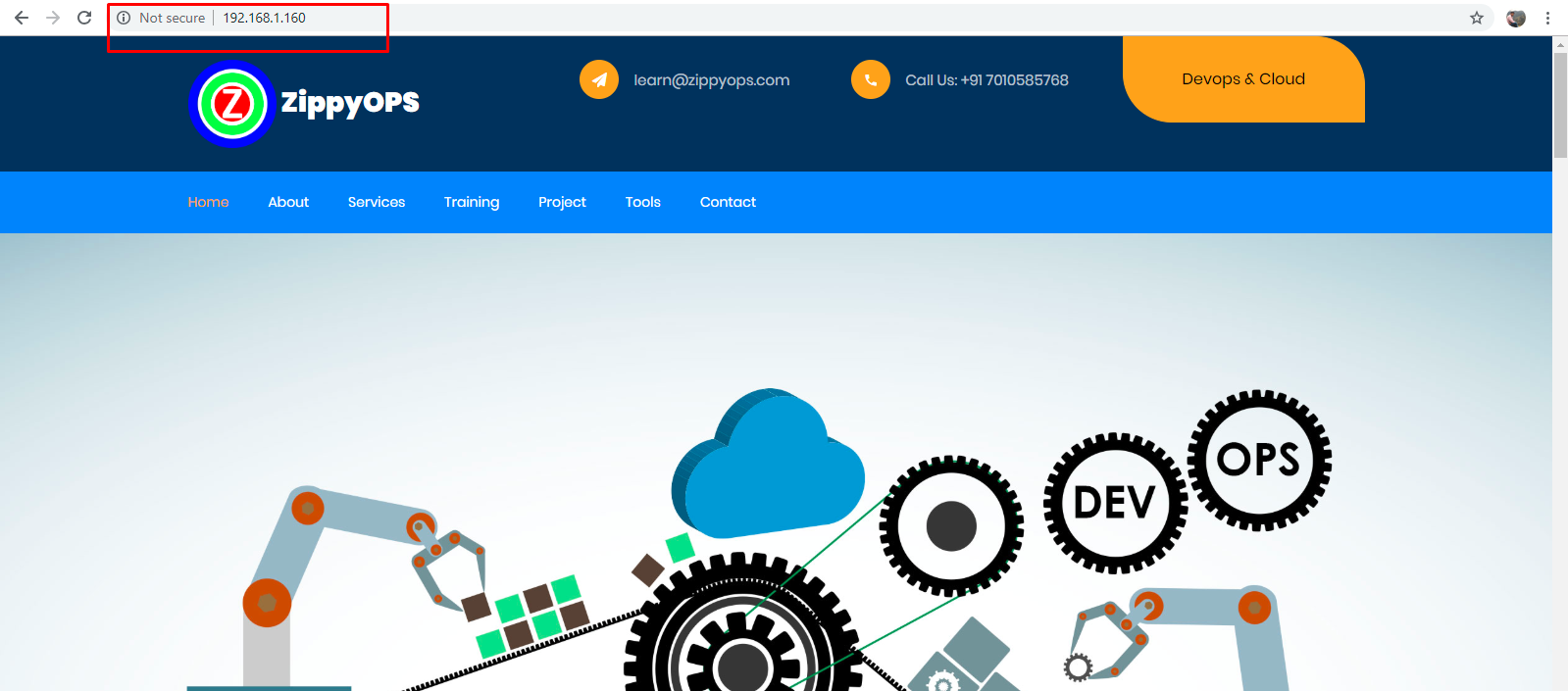
## OUTPUT





**Add html file in nfs server**





Now the pods are displayed in external ip 192.168.1.160 whenever there is changes given to nfs server it will be automatically updated in the running pods

Reference

**For high availability k8s** => <https://blog.inkubate.io/install-and-configure-a-multi-master-kubernetes-cluster-with-kubeadm/>

**For metalLB** => <https://medium.com/@JockDaRock/metalloadbalancer-kubernetes-on-prem-baremetal-loadbalancing-101455c3ed48>

<https://medium.com/@JockDaRock/kubernetes-metal-lb-for-on-prem-baremetal-cluster-in-10-minutes-c2eaeb3fe813>

**for NFS persistent volume** => <https://www.youtube.com/watch?v=to14wmNmRCI&t=785s>

<https://github.com/justmeandopensource/kubernetes> => very useful