Project Report

On

Automation Hardening of Kubernetes Cluster

Submitted in fulfilment for the award of

Post Graduate Diploma in IT Infrastructure System & Security (PG-DITISS) from **CDAC ACTS (Pune)**

Guided By:

Mr. Lavish Jhamb

Presented By:

Sawant Ashutosh Santosh	PRN: 210540185010
Ghuge Kirtiraj Ashok	PRN: 210540185018
Mahajan Pranav Prashant	PRN: 210540185035
Kukde Atish Vinod	PRN: 210540185011
Attar Mohsin Daut	PRN: 210540185026

Centre of Development of Advanced Computing (C-DAC), Pune

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Sawant Ashutosh Santosh	**PRN: 210540185010**
Ghuge Kirtiraj Ashok	**PRN: 210540185018**
Mahajan Pranav Prashant	**PRN: 210540185035**
Kukde Atish Vinod	**PRN: 210540185011**
Attar Mohsin Daut	**PRN: 210540185026**

Have successfully completed their project on

“Automation Hardening of Kubernetes Cluster”

Under the guidance of

Mr. Lavish Jhamb

Project Guide	**Project Supervisor**
Mr. Lavish Jhamb	Mr. Lavish Jhamb

HOD ACTS
Mr. Gaur Sunder

ACKNOWLEDGEMENT

This project “Automation Hardening of Kubernetes Cluster” was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We all are very glad to mention the name of *Mr. Lavish Jhamb* for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

We are highly grateful to *Ms. Risha P.R.* (Manager, ACTS training Centre), C-DAC, for her guidance and support whenever necessary while doing this course Post Graduate Diploma in *IT Infrastructure, Systems and Security (PG-DITISS)* through C-DAC ACTS, Pune.

Our most heartfelt thanks go to *Mr. Zubeen Sayyad* (Course Coordinator, PG-*DITISS*) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in C- DAC ACTS, Pune.

Sincerely,

Sawant Ashutosh
Ghuge Kirtiraj
Mahajan Pranav
Kukde Atish
Attar Mohsin

ABSTRACT

Kubernetes is a vendor-agnostic cluster and container management tool, open-sourced by Google in 2014. It provides a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts". Above all, this lowers the cost of cloud computing expenses and simplifies operations and architecture. A container is a mini-virtual machine. It is small, as it does not have device drivers and all the other components of a regular virtual machine. Docker is by far the most popular container and it is written in Linux.

But there is an inherent problem with containers, just like there is with virtual machines. That is the need to keep track of them. When public cloud companies bill you for CPU time or storage then you need to make sure you do not have any orphaned machines spinning out there doing nothing. Plus, there is the need to automatically spin up more when a machine needs more memory, CPU, or storage, as well as shut them down when the load lightens. Orchestration tackles these problems. This is where Kubernetes comes in.

So, in this project scanning and hardening of Kubernetes clusters is done using Bash Shell Scripting with the help of CIS Kubernetes benchmarks 2020. This Guidelines are intended for system and application administrators, security specialists, auditors, help desk, and platform deployment personnel who plan to develop, deploy, assess, or secure solutions that incorporate Kubernetes.

TABLE OF CONTENTS

1.INTRODUCTION AND OVERVIEW OF PROJECT

1. Purpose

The Hardening of Kubernetes clustering with the help of the CIS Benchmark of Kubernetes 2020 using a bash script for scanning and security.

2. Aims and Objective

To setup Kubernetes cluster using Bash shell script. Also provide proper and appropriate scanning and hardening of the deployed cluster. This project also aims to set the best-practice of cybersecurity standards for Kubernetes clustering.

Objectives:

- Automate cluster deployment
- Automation of scanning and hardening for cluster deployment

2.OVERALL DESCRIPTION

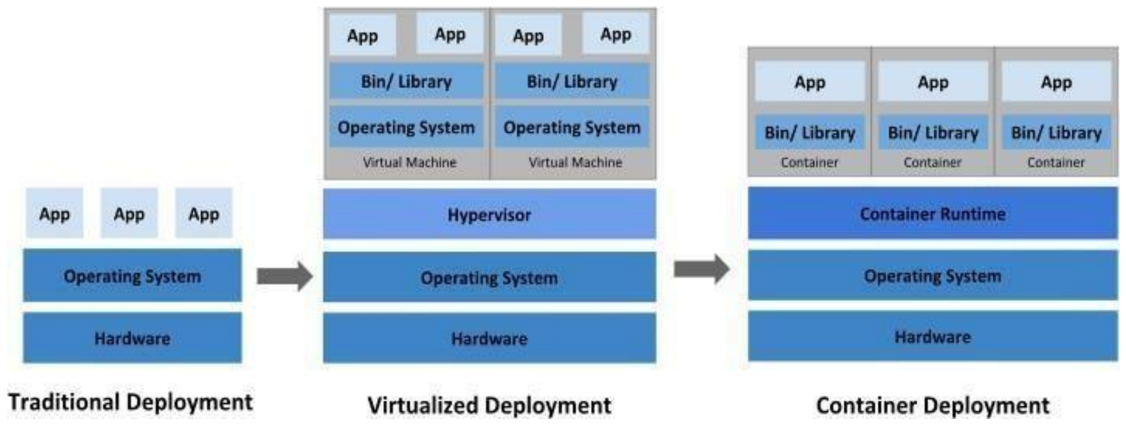
2.1. Introduction:

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

The name Kubernetes originates from Greek, meaning helmsman or pilot. Google open-sourced the Kubernetes project in 2014. Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale, combined with best-of-breed ideas and practices from the community.

Kubernetes is itself not a Platform as a Service (PaaS) tool, but it serves as more of a basic framework, allowing users to choose the types of application frameworks, languages, monitoring and logging tools, and other tools of their choice. In this way, Kubernetes can be used as the basis for a complete PaaS to run on top of; this is the architecture chosen by the OpenShift Origin open-source project in its latest release.

Let's take a look at why Kubernetes is so useful by going back in time:



Traditional deployment era: Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues. For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform. A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers.

Virtualized deployment era: As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.

Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines.

Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

Container deployment era: Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM, a container has its own filesystem, CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

Difference between Kubernetes and Docker Swarm:

Features	Kubernetes	Docker Swarm
Installation Cluster Config	Setup is very complicated, but once installed cluster is robust.	Installation is very simple, but the cluster is not robust.
GUI	GUI is the Kubernetes Dashboard	There is no GUI.
Scalability	Highly scalable and scales fast.	Highly scalable
Auto-scaling	Kubernetes can do auto-scaling.	Docker swarm cannot do auto-scaling
Load Balancing	Manual intervention needed f load balancing traffic between different containers and pods.	Docker swarm does auto load balancing of traffic between containers in the cluster.
Rolling Updated & Rollbacks	Can deploy rolling updates an does automatic rollbacks.	Can deploy rolling updates, but n automatic rollback.
DATA Volume	Can share storage volumes on with the other containers in t same pod.	Can share storage volumes with another container.
Logging Monitoring	In-built tools for logging a monitoring.	3rd party tools like ELK stack should be used for logging and monitoring.

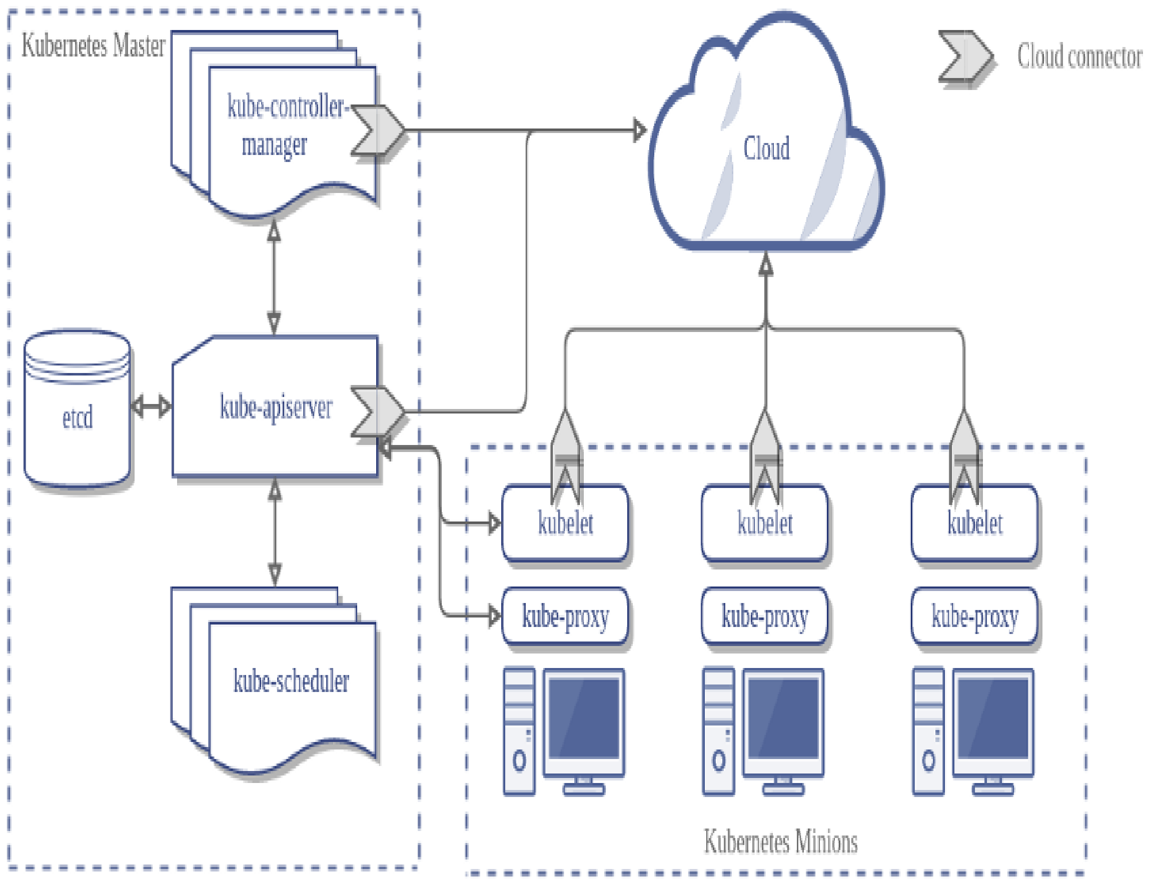
2.2 Architecture

Kubernetes introduces a lot of vocabulary to describe how your application is organized. We'll start from the smallest layer and work our way up.

Master server

This is the main entry point for administrators and users to manage the various nodes. Operations are issued to it either through HTTP calls or connecting to the machine and running command-line scripts.

The master server consists of various components including a kube-apiserver, an etcd storage, a kube-controller-manager, a cloud-controller-manager, a kube-scheduler, and a DNS server for Kubernetes services

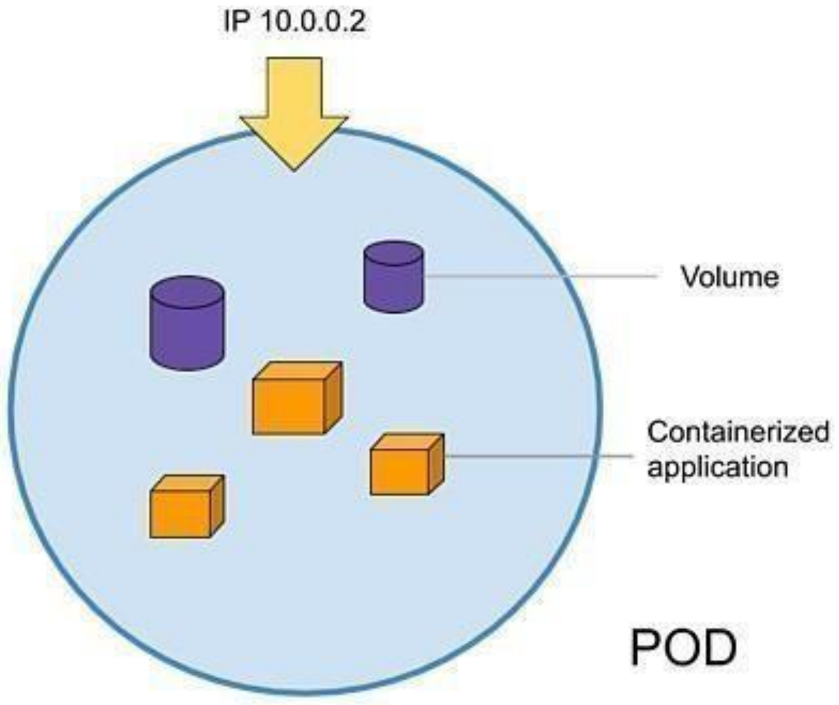


Nodes

A Kubernetes node manages and runs pods; it's the machine (whether virtualized or physical) that performs the given work. Just as pods collect individual containers that operate together, a node collects entire pods that function together. When you're operating at scale, you want to be able to hand work over to a node whose pods are free to take it.

Pods

A Kubernetes pod is a group of containers, and is the smallest unit that Kubernetes administers. Pods have a single IP address that is applied to every container within the pod. Containers in a pod share the same resources such as memory and storage. This allows the individual Linux containers inside a pod to be treated collectively as a single application, as if all the containerized processes were running together on the same host in more traditional workloads. It's quite common to have a pod with only a single container, when the application or service is a single process that needs to run. But when things get more complicated, and multiple processes need to work together using the same shared data volumes for correct operation, multi-container pods ease deployment configuration compared to setting up shared resources between containers on your own.



For example, if you were working on an image-processing service that created GIFs, one pod might have several containers working together to resize images. The primary container might be running the non-blocking microservice application taking in requests, and then one or more auxiliary (side-car) containers running batched background processes or cleaning up data artifacts in the storage volume as part of managing overall application performance.

Deployments

Kubernetes deployments define the scale at which you want to run your application by letting you set the details of how you would like pods replicated on your Kubernetes nodes. Deployments describe the number of desired identical pod replicas to run and the preferred update strategy used when updating the deployment. Kubernetes will track pod health, and will remove or add pods as needed to bring your application deployment to the desired state.

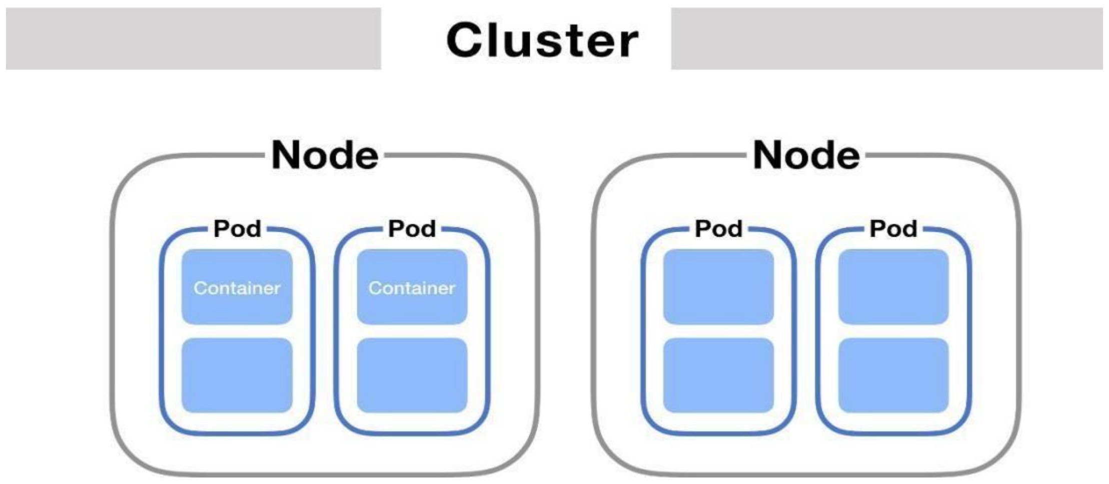
Services

The lifetime of an individual pod cannot be relied upon; everything from their IP addresses to their very existence are prone to change. In fact, within the DevOps community, there’s the notion of treating servers as either “pets” or “cattle.” A pet is something you take special care of, whereas cows are viewed as somewhat more expendable. In the same vein, Kubernetes doesn’t treat its pods as unique, long-running instances; if a pod encounters an issue and dies, it’s Kubernetes’ job to replace it so that the application doesn’t experience any downtime.

A service is an abstraction over the pods, and essentially, the only interface the various application consumers interact with. As pods are replaced, their internal names and IPs might change. A service exposes a single machine name or IP address mapped to pods whose underlying names and numbers are unreliable. A service ensures that, to the outside network, everything appears to be unchanged.

Cluster

A cluster is all of the above components put together as a single unit. A cluster consists of at least one cluster master and multiple worker machines called nodes. These master and node machines run the Kubernetes cluster orchestration system.



Kubernetes components

With a general idea of how Kubernetes is assembled, it's time to take a look at the various software components that make sure everything runs smoothly. Both the master server and individual worker nodes have three main components each.

Master server components

API Server

The API server exposes a REST interface to the Kubernetes cluster. All operations against pods, services, and so forth, are executed programmatically by communicating with the endpoints provided by it. It’s a connector between all the Kubernetes components and mediates all interactions between clients and the API objects stored in etcd. The APIs are exposed and managed by the server, the characteristics of those API requests must be described so that the client and server know how to communicate.

Scheduler

The scheduler is responsible for assigning work to the various nodes. It keeps watch over the resource capacity and ensures that a worker node's performance is within an appropriate threshold. It is a simple algorithm that defines the priority to dispatch and is responsible for scheduling pods into nodes. It is continuously scanning the API server (with watch protocol) for Pods which don't have a node Name and are eligible for scheduling.

Controller manager

The controller-manager is responsible for making sure that the shared state of the cluster is operating as expected. More accurately, the controller manager oversees various controllers which respond to events (e.g., if a node goes down). Controller manager is a collection of control loops rolled up into one binary. It creates and updates the Kubernetes internal information.

Worker node components

Kubelet

The kubelet is the primary "node agent" that runs on each node. It can register the node with the apiserver using one of: the hostname; a flag to override the hostname; or specific logic for a cloud provider.

The kubelet works in terms of a PodSpec. A PodSpec is a YAML or JSON object that describes a pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms (primarily through the apiserver) 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.

Other than from a PodSpec from the apiserver, there are three ways that a container manifest can be provided to the Kubelet.

A Kubelet tracks the state of a pod to ensure that all the containers are running. It provides a heartbeat message every few seconds to the master server. If a replication controller does not receive that message, the node is marked as unhealthy.

Kube proxy

The Kube proxy routes traffic coming into a node from the service. It forwards requests for work to the correct containers. 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.

The Kubernetes network proxy runs on each node. This reflects services as defined in the Kubernetes API on each node and can-do simple TCP, UDP, and SCTP stream forwarding or round robin TCP, UDP, and SCTP forwarding across a set of backends.

Service cluster IPs and ports are currently found through Docker-links-compatible environment variables specifying ports opened by the service proxy. There is an optional addon that provides cluster DNS for these cluster IPs. The user must create a service with the apiserver API to configure the proxy.

3. SYSTEM REQUIREMENTS

Base Machine Software

Requirements

- VM Ware Workstation Pro 15 on Windows10

Hardware Requirements

- RAM 32 GB
- Processor i7/ Ryzen 5 or above
- Hard disk 20GB

Master Node

Software Requirements

- CentOS 7

Hardware Requirements

- RAM 4GB
- Number of vCPU’s: 4

Worker Node

Software Requirements

- CentOS 7 minimal

Hardware Requirements

- RAM 1GB
- Number of vCPU’s: 1

4. INSTALLATION AND SCREENSHOTS

Kubernetes is a container orchestration system that manages containers at scale. Initially developed by Google based on its experience running containers in production, Kubernetes is open source and actively developed by a community around the world.

Kubeadm automates the installation and configuration of Kubernetes components such as the API server, Controller Manager, and Kube DNS. It does not, however, create users or handle the installation of operating-system-level dependencies and their configuration. Using bash shell script, we create cluster between master node and worker nodes.

In this guide, you will set up a Kubernetes cluster from Kubeadm and token using bash shell script.

Goals- cluster will include the following physical resources:

One master node

The master node (a node in Kubernetes refers to a server) is responsible for managing the state of the cluster. It runs Etcd, which stores cluster data among components that schedule workloads to worker nodes.

Worker nodes

Worker nodes are the servers where your workloads (i.e., containerized applications and services) will run. A worker will continue to run your workload once they're assigned to it, even if the master goes down once scheduling is complete. A cluster's capacity can be increased by adding workers.

After completing this cluster setup, you will have a cluster ready to run containerized applications, provided that the servers in the cluster have sufficient CPU and RAM resources for your applications to consume. The cluster itself will consume around 300-500MB of memory and 10% of CPU on each node.

Prerequisites

On all nodes (master, worker)

Disable swap as its not required and sometimes causes performance issues so its suggested to disable it.

```
# swapoff –a
```

Also comment swap entry in /etc/fstab to disable swap permanently.

On master

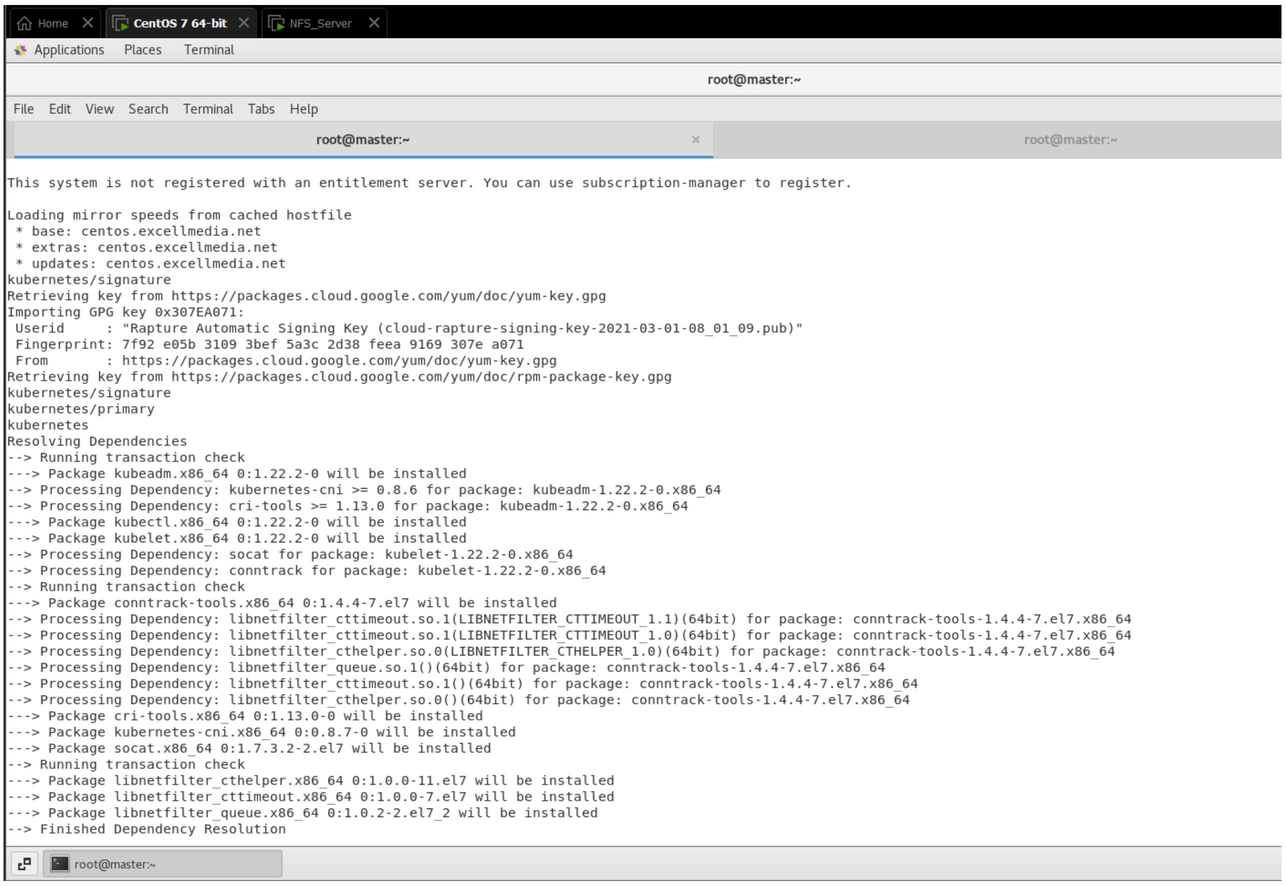
BR netfilter module is required to enable transparent masquerading and to facilitate Virtual Extensible LAN (VxLAN) traffic for communication between Kubernetes pods across the cluster. For that we are enabling br_netfilter kernel module.

```
###Enable br_netfilter kernel module
modprobe br_netfilter
echo '1' > /proc/sys/net/bridge/bridge-nf-call-ip6tables
echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
touch /etc/sysctl.d/k8s.conf
echo 'net.bridge.bridge-nf-call-ip6tables = 1' > /etc/sysctl.d/k8s.conf
echo 'net.bridge.bridge-nf-call-iptables = 1' >> /etc/sysctl.d/k8s.conf
```

Step 1 — Add kubernetes repository on each node.

In this section, we create a kubernetes repository on each machine. Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery.

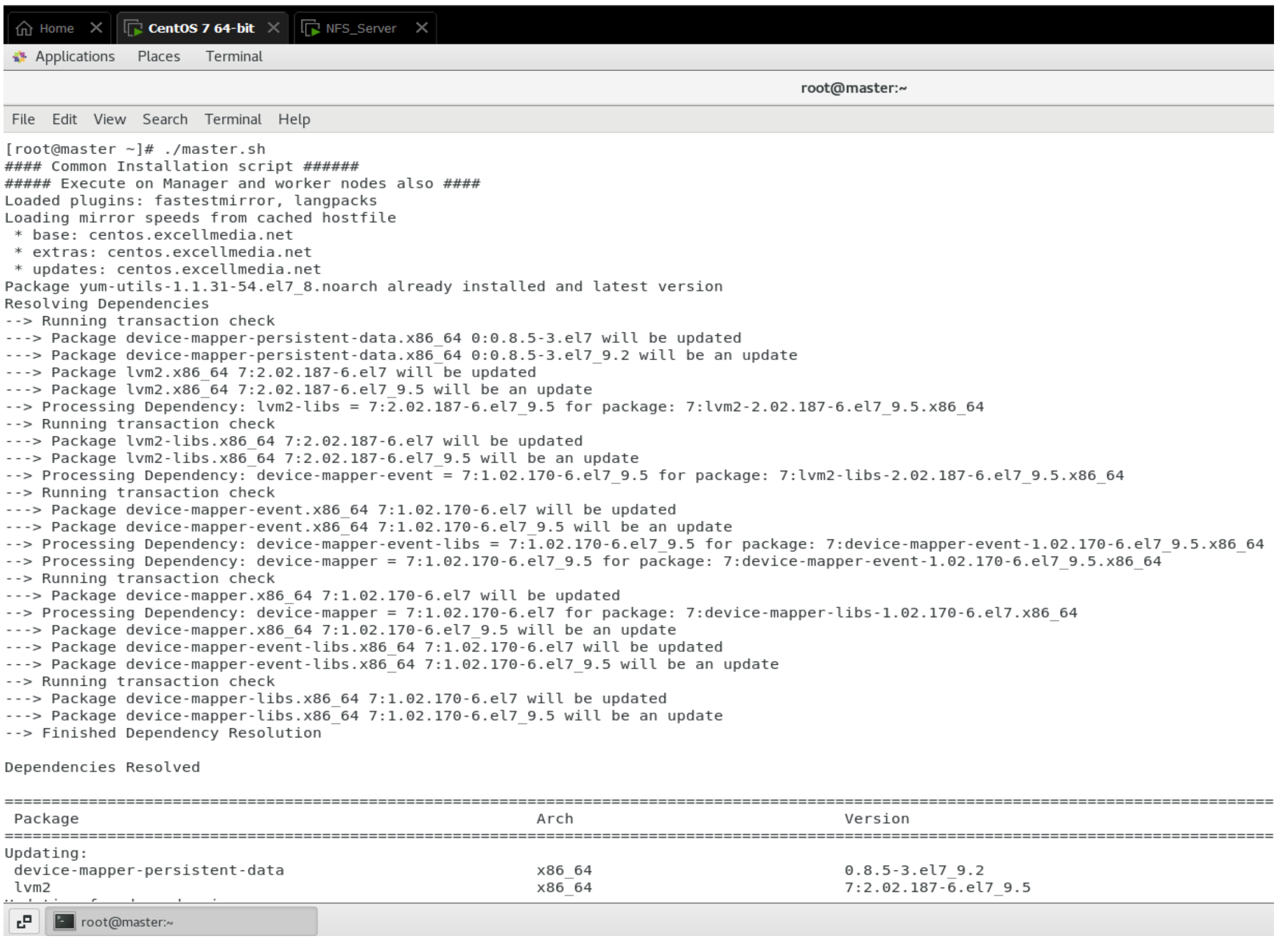
```
###Add kubernetes repository
cd /etc/yum.repos.d
touch kubernetes.repo
echo '[kubernetes]' > kubernetes.repo
echo 'name=Kubernetes' >> kubernetes.repo
echo 'baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64' >>
kubernetes.repo
echo 'enabled=1' >> kubernetes.repo
echo 'gpgcheck=1' >> kubernetes.repo
echo 'repo_gpgcheck=1' >> kubernetes.repo
echo 'gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg' >> kubernetes.repo
```



```
This system is not registered with an entitlement server. You can use subscription-manager to register.

Loading mirror speeds from cached hostfile
 * base: centos.excellmedia.net
 * extras: centos.excellmedia.net
 * updates: centos.excellmedia.net
kubernetes/signature
Retrieving key from https://packages.cloud.google.com/yum/doc/yum-key.gpg
Importing GPG key 0x307EA071:
 Userid     : "Rapture Automatic Signing Key (cloud-rapture-signing-key-2021-03-01-08_01_09.pub)"
 Fingerprint: 7f92 e05b 3109 3bef 5a3c 2d38 feea 9169 307e a071
 From       : https://packages.cloud.google.com/yum/doc/yum-key.gpg
Retrieving key from https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
kubernetes/signature
kubernetes/primary
kubernetes
Resolving Dependencies
--> Running transaction check
---> Package kubeadm.x86_64 0:1.22.2-0 will be installed
--> Processing Dependency: kubernetes-cni >= 0.8.6 for package: kubeadm-1.22.2-0.x86_64
--> Processing Dependency: cri-tools >= 1.13.0 for package: kubeadm-1.22.2-0.x86_64
---> Package kubectl.x86_64 0:1.22.2-0 will be installed
---> Package kubelet.x86_64 0:1.22.2-0 will be installed
--> Processing Dependency: socat for package: kubelet-1.22.2-0.x86_64
--> Processing Dependency: conntrack for package: kubelet-1.22.2-0.x86_64
--> Running transaction check
---> Package conntrack-tools.x86_64 0:1.4.4-7.el7 will be installed
--> Processing Dependency: libnetfilter_cttimeout.so.1(LIBNETFILTER_CTTIMEOUT_1.1)(64bit) for package: conntrack-tools-1.4.4-7.el7.x86_64
--> Processing Dependency: libnetfilter_cttimeout.so.1(LIBNETFILTER_CTTIMEOUT_1.0)(64bit) for package: conntrack-tools-1.4.4-7.el7.x86_64
--> Processing Dependency: libnetfilter_cthelper.so.0(LIBNETFILTER_CTHELPER_1.0)(64bit) for package: conntrack-tools-1.4.4-7.el7.x86_64
--> Processing Dependency: libnetfilter_queue.so.1()(64bit) for package: conntrack-tools-1.4.4-7.el7.x86_64
--> Processing Dependency: libnetfilter_cttimeout.so.1()(64bit) for package: conntrack-tools-1.4.4-7.el7.x86_64
--> Processing Dependency: libnetfilter_cthelper.so.0()(64bit) for package: conntrack-tools-1.4.4-7.el7.x86_64
---> Package cri-tools.x86_64 0:1.13.0-0 will be installed
---> Package kubernetes-cni.x86_64 0:0.8.7-0 will be installed
---> Package socat.x86_64 0:1.7.3.2-2.el7 will be installed
--> Running transaction check
---> Package libnetfilter_cthelper.x86_64 0:1.0.0-11.el7 will be installed
---> Package libnetfilter_cttimeout.x86_64 0:1.0.0-7.el7 will be installed
---> Package libnetfilter_queue.x86_64 0:1.0.2-2.el7_2 will be installed
--> Finished Dependency Resolution
```

Step 2 — Installing of yum_utils, device_mapper_persistent_data, lvm2In this section, you will install the yum_utils, device_mapper_persistent_data and lum2. The yum_utils is a set of Go libraries that provide low-level, kubernetes-independent packages supplementing the Go standard libs. The Device Mapper (device_mapper_persistent_data) is a kernel-based framework that underpins many advanced volume management technologies on Linux and CSI LVM Provisioner utilizes local storage of Kubernetes nodes to provide persistent storage for pods. Underneath it creates a LVM logical volume on the local disks



```
[root@master ~]# ./master.sh
#### Common Installation script ######
##### Execute on Manager and worker nodes also ####
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: centos.excellmedia.net
 * extras: centos.excellmedia.net
 * updates: centos.excellmedia.net
Package yum-utils-1.1.31-54.el7_8.noarch already installed and latest version
Resolving Dependencies
--> Running transaction check
---> Package device-mapper-persistent-data.x86_64 0:0.8.5-3.el7 will be updated
---> Package device-mapper-persistent-data.x86_64 0:0.8.5-3.el7_9.2 will be an update
---> Package lvm2.x86_64 7:2.02.187-6.el7 will be updated
---> Package lvm2.x86_64 7:2.02.187-6.el7_9.5 will be an update
--> Processing Dependency: lvm2-libs = 7:2.02.187-6.el7_9.5 for package: 7:lvm2-2.02.187-6.el7_9.5.x86_64
--> Running transaction check
---> Package lvm2-libs.x86_64 7:2.02.187-6.el7 will be updated
---> Package lvm2-libs.x86_64 7:2.02.187-6.el7_9.5 will be an update
--> Processing Dependency: device-mapper-event = 7:1.02.170-6.el7_9.5 for package: 7:lvm2-libs-2.02.187-6.el7_9.5.x86_64
--> Running transaction check
---> Package device-mapper-event.x86_64 7:1.02.170-6.el7 will be updated
---> Package device-mapper-event.x86_64 7:1.02.170-6.el7_9.5 will be an update
--> Processing Dependency: device-mapper-event-libs = 7:1.02.170-6.el7_9.5 for package: 7:device-mapper-event-1.02.170-6.el7_9.5.x86_64
--> Processing Dependency: device-mapper = 7:1.02.170-6.el7_9.5 for package: 7:device-mapper-event-1.02.170-6.el7_9.5.x86_64
--> Running transaction check
---> Package device-mapper.x86_64 7:1.02.170-6.el7 will be updated
--> Processing Dependency: device-mapper = 7:1.02.170-6.el7 for package: 7:device-mapper-libs-1.02.170-6.el7.x86_64
---> Package device-mapper.x86_64 7:1.02.170-6.el7_9.5 will be an update
---> Package device-mapper-event-libs.x86_64 7:1.02.170-6.el7 will be updated
---> Package device-mapper-event-libs.x86_64 7:1.02.170-6.el7_9.5 will be an update
--> Running transaction check
---> Package device-mapper-libs.x86_64 7:1.02.170-6.el7 will be updated
---> Package device-mapper-libs.x86_64 7:1.02.170-6.el7_9.5 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

=========================================================================================================================================
 Package                                                Arch                              Version
=========================================================================================================================================
Updating:
 device-mapper-persistent-data                          x86_64                            0.8.5-3.el7_9.2
 lvm2                                                   x86_64                            7:2.02.187-6.el7_9.5
```

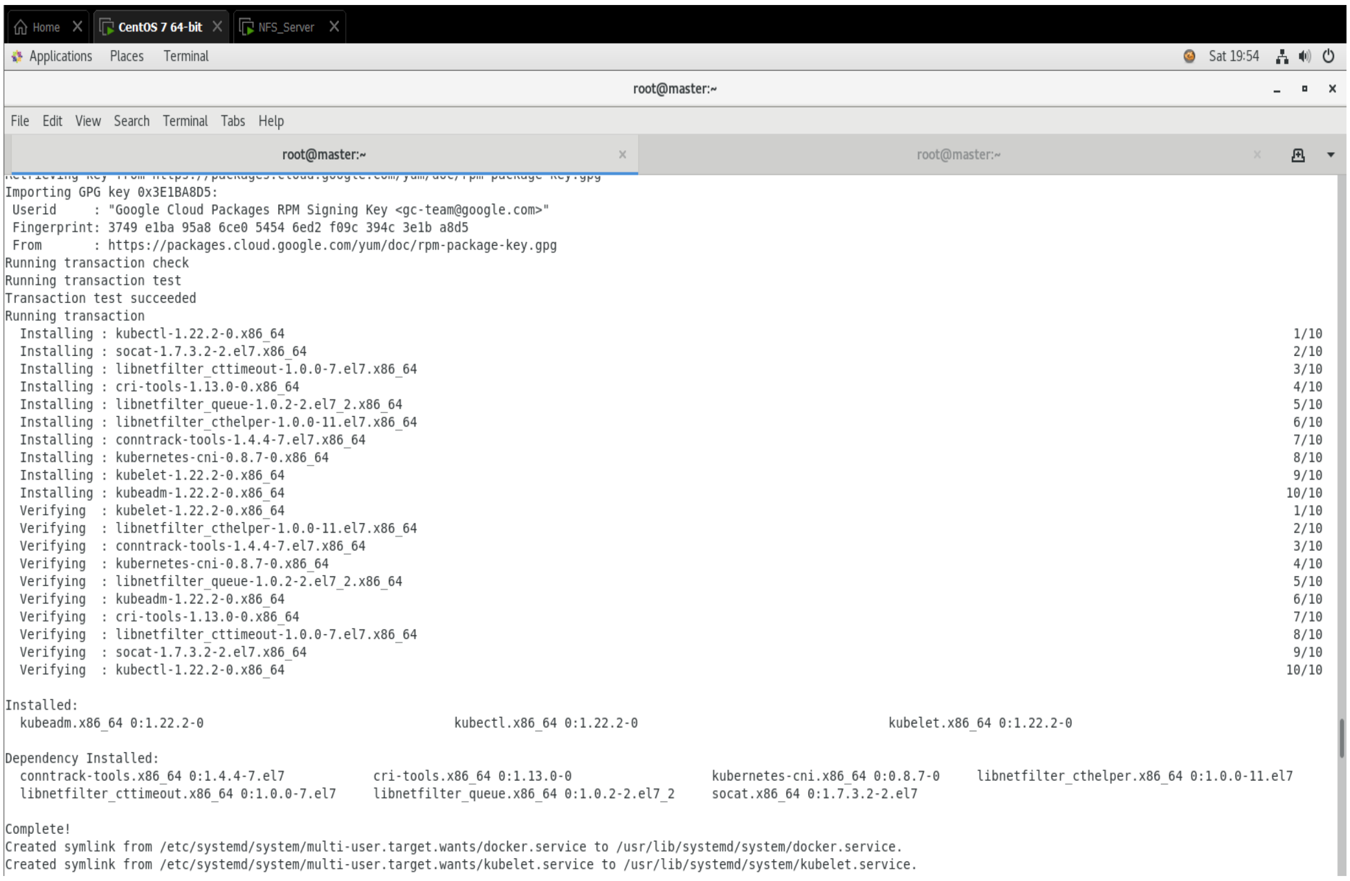
Step 3 — Installing kubelet, kubectl and Kubeadm

kubelet - a system service/program that runs on all nodes and handle node-level operations.

kubeadm - a CLI tool that will install and configure the various components of a cluster in a standard way.

kubectl - a CLI tool used for issuing commands to the cluster through it

apiserver.



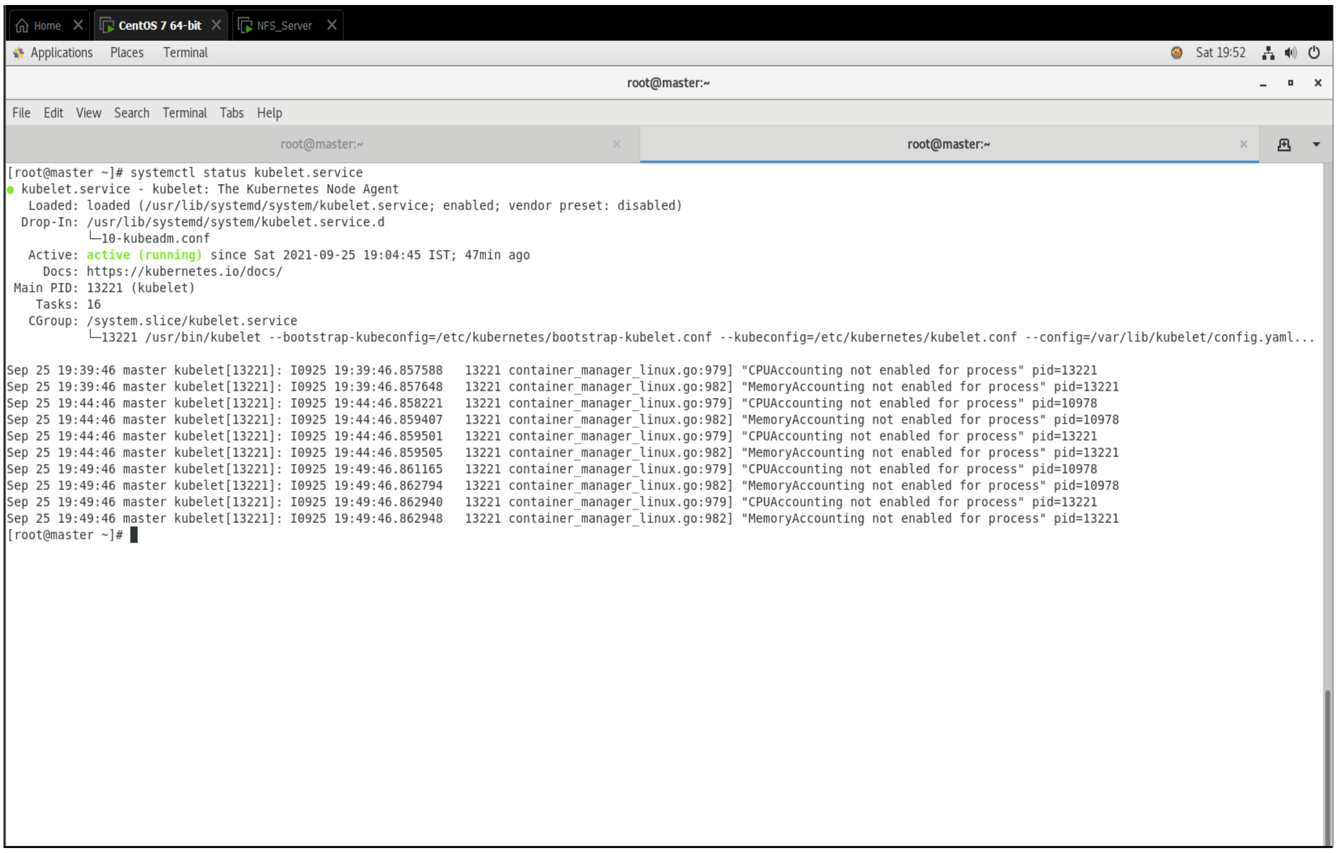
```
Importing GPG key 0x3E1BA8D5:
 Userid     : "Google Cloud Packages RPM Signing Key <gc-team@google.com>"
 Fingerprint: 3749 e1ba 95a8 6ce0 5454 6ed2 f09c 394c 3e1b a8d5
 From       : https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : kubectl-1.22.2-0.x86_64                                                  1/10
  Installing : socat-1.7.3.2-2.el7.x86_64                                               2/10
  Installing : libnetfilter_cttimeout-1.0.0-7.el7.x86_64                                3/10
  Installing : cri-tools-1.13.0-0.x86_64                                                4/10
  Installing : libnetfilter_queue-1.0.2-2.el7_2.x86_64                                  5/10
  Installing : libnetfilter_cthelper-1.0.0-11.el7.x86_64                                6/10
  Installing : conntrack-tools-1.4.4-7.el7.x86_64                                       7/10
  Installing : kubernetes-cni-0.8.7-0.x86_64                                            8/10
  Installing : kubelet-1.22.2-0.x86_64                                                  9/10
  Installing : kubeadm-1.22.2-0.x86_64                                                 10/10
  Verifying  : kubelet-1.22.2-0.x86_64                                                  1/10
  Verifying  : libnetfilter_cthelper-1.0.0-11.el7.x86_64                                2/10
  Verifying  : conntrack-tools-1.4.4-7.el7.x86_64                                       3/10
  Verifying  : kubernetes-cni-0.8.7-0.x86_64                                            4/10
  Verifying  : libnetfilter_queue-1.0.2-2.el7_2.x86_64                                  5/10
  Verifying  : kubeadm-1.22.2-0.x86_64                                                  6/10
  Verifying  : cri-tools-1.13.0-0.x86_64                                                7/10
  Verifying  : libnetfilter_cttimeout-1.0.0-7.el7.x86_64                                8/10
  Verifying  : socat-1.7.3.2-2.el7.x86_64                                               9/10
  Verifying  : kubectl-1.22.2-0.x86_64                                                 10/10

Installed:
  kubeadm.x86_64 0:1.22.2-0                kubectl.x86_64 0:1.22.2-0                kubelet.x86_64 0:1.22.2-0

Dependency Installed:
  conntrack-tools.x86_64 0:1.4.4-7.el7          cri-tools.x86_64 0:1.13.0-0                kubernetes-cni.x86_64 0:0.8.7-0      libnetfilter_cthelper.x86_64 0:1.0.0-11.el7
  libnetfilter_cttimeout.x86_64 0:1.0.0-7.el7   libnetfilter_queue.x86_64 0:1.0.2-2.el7_2  socat.x86_64 0:1.7.3.2-2.el7

Complete!
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to /usr/lib/systemd/system/kubelet.service.
```

Status of installed kubelet service:

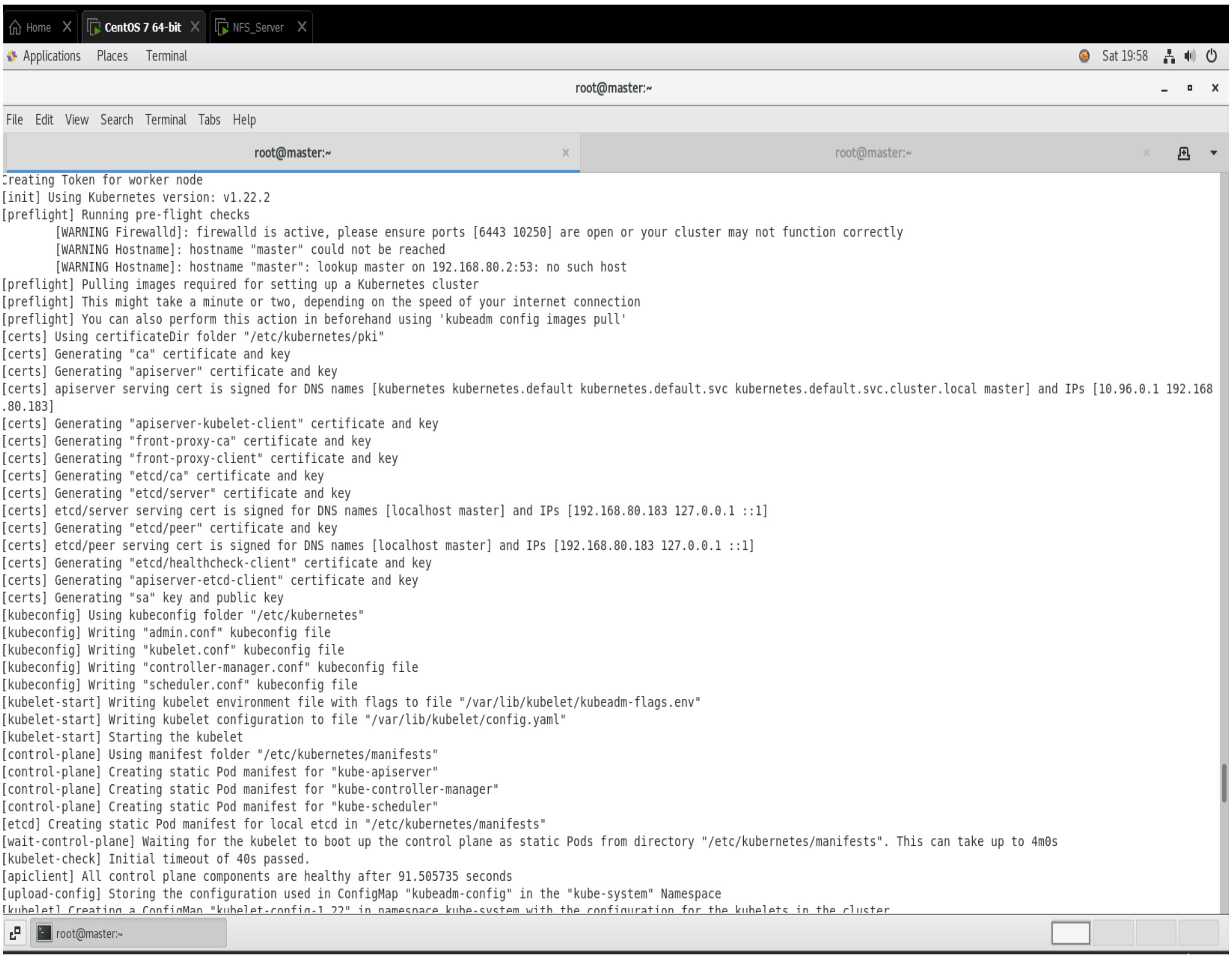


```
[root@master ~]# systemctl status kubelet.service
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/kubelet.service.d
           └─10-kubeadm.conf
   Active: active (running) since Sat 2021-09-25 19:04:45 IST; 47min ago
     Docs: https://kubernetes.io/docs/
 Main PID: 13221 (kubelet)
    Tasks: 16
   CGroup: /system.slice/kubelet.service
           └─13221 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/lib/kubelet/config.yaml...

Sep 25 19:39:46 master kubelet[13221]: I0925 19:39:46.857588   13221 container_manager_linux.go:979] "CPUAccounting not enabled for process" pid=13221
Sep 25 19:39:46 master kubelet[13221]: I0925 19:39:46.857648   13221 container_manager_linux.go:982] "MemoryAccounting not enabled for process" pid=13221
Sep 25 19:44:46 master kubelet[13221]: I0925 19:44:46.858221   13221 container_manager_linux.go:979] "CPUAccounting not enabled for process" pid=10978
Sep 25 19:44:46 master kubelet[13221]: I0925 19:44:46.859407   13221 container_manager_linux.go:982] "MemoryAccounting not enabled for process" pid=10978
Sep 25 19:44:46 master kubelet[13221]: I0925 19:44:46.859501   13221 container_manager_linux.go:979] "CPUAccounting not enabled for process" pid=13221
Sep 25 19:44:46 master kubelet[13221]: I0925 19:44:46.859505   13221 container_manager_linux.go:982] "MemoryAccounting not enabled for process" pid=13221
Sep 25 19:49:46 master kubelet[13221]: I0925 19:49:46.861165   13221 container_manager_linux.go:979] "CPUAccounting not enabled for process" pid=10978
Sep 25 19:49:46 master kubelet[13221]: I0925 19:49:46.862794   13221 container_manager_linux.go:982] "MemoryAccounting not enabled for process" pid=10978
Sep 25 19:49:46 master kubelet[13221]: I0925 19:49:46.862940   13221 container_manager_linux.go:979] "CPUAccounting not enabled for process" pid=13221
Sep 25 19:49:46 master kubelet[13221]: I0925 19:49:46.862948   13221 container_manager_linux.go:982] "MemoryAccounting not enabled for process" pid=13221
[root@master ~]#
```

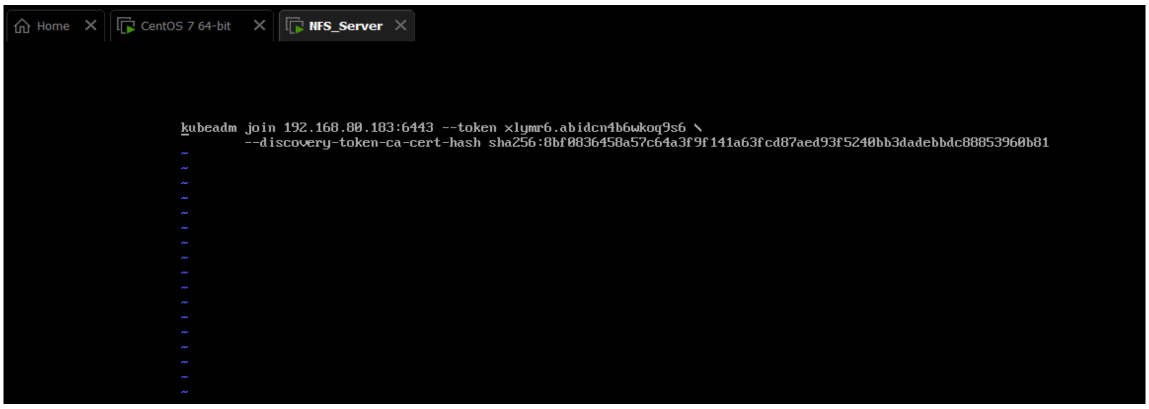
Step 4 — Creating token for worker node

In this section, you will set up the worker node using token with help of “kubeadm init” command.



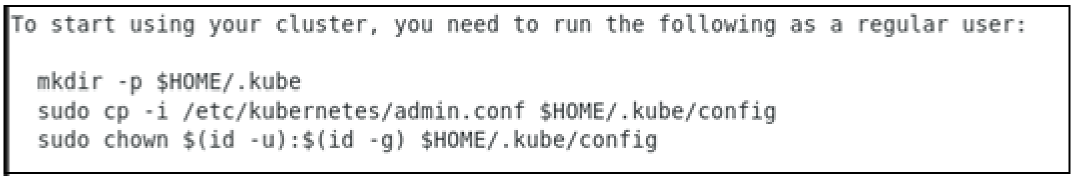
```
Creating Token for worker node
[init] Using Kubernetes version: v1.22.2
[preflight] Running pre-flight checks
        [WARNING Firewalld]: firewalld is active, please ensure ports [6443 10250] are open or your cluster may not function correctly
        [WARNING Hostname]: hostname "master" could not be reached
        [WARNING Hostname]: hostname "master": lookup master on 192.168.80.2:53: no such host
[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 'kubeadm 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.svc kubernetes.default.svc.cluster.local master] and IPs [10.96.0.1 192.168
.80.183]
[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 master] and IPs [192.168.80.183 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [localhost master] and IPs [192.168.80.183 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s
[kubelet-check] Initial timeout of 40s passed.
[apiclient] All control plane components are healthy after 91.505735 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.22" in namespace kube-system with the configuration for the kubelets in the cluster
```

Token in worker node:



```
kubeadm join 192.168.80.183:6443 --token xlymr6.abidcn4b6wkoq9s6 \
        --discovery-token-ca-cert-hash sha256:8bf0836458a57c64a3f9f141a63fcd87aed93f5240bb3dadebbdc88853960b81
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Step 5 — Making directory kube and changing ownership

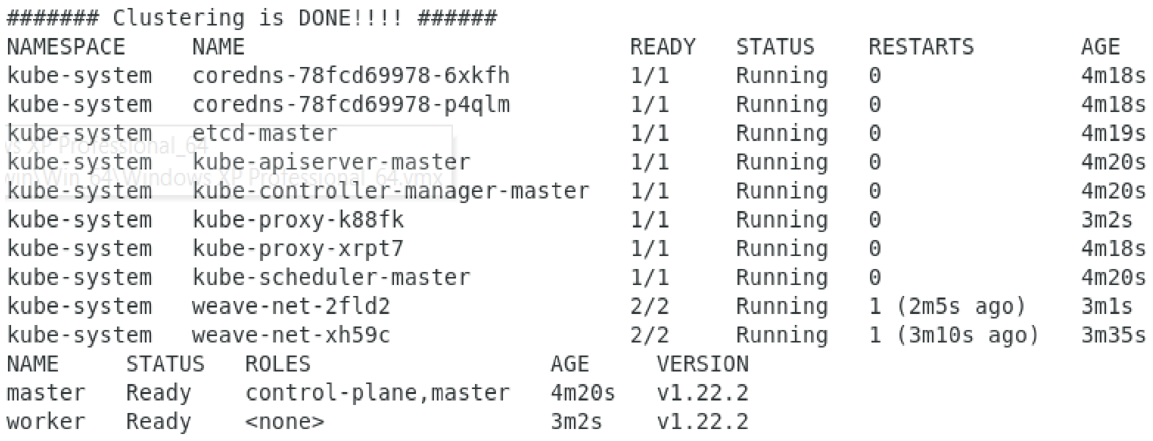


```
To start using your cluster, you need to run the following as a regular user:

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

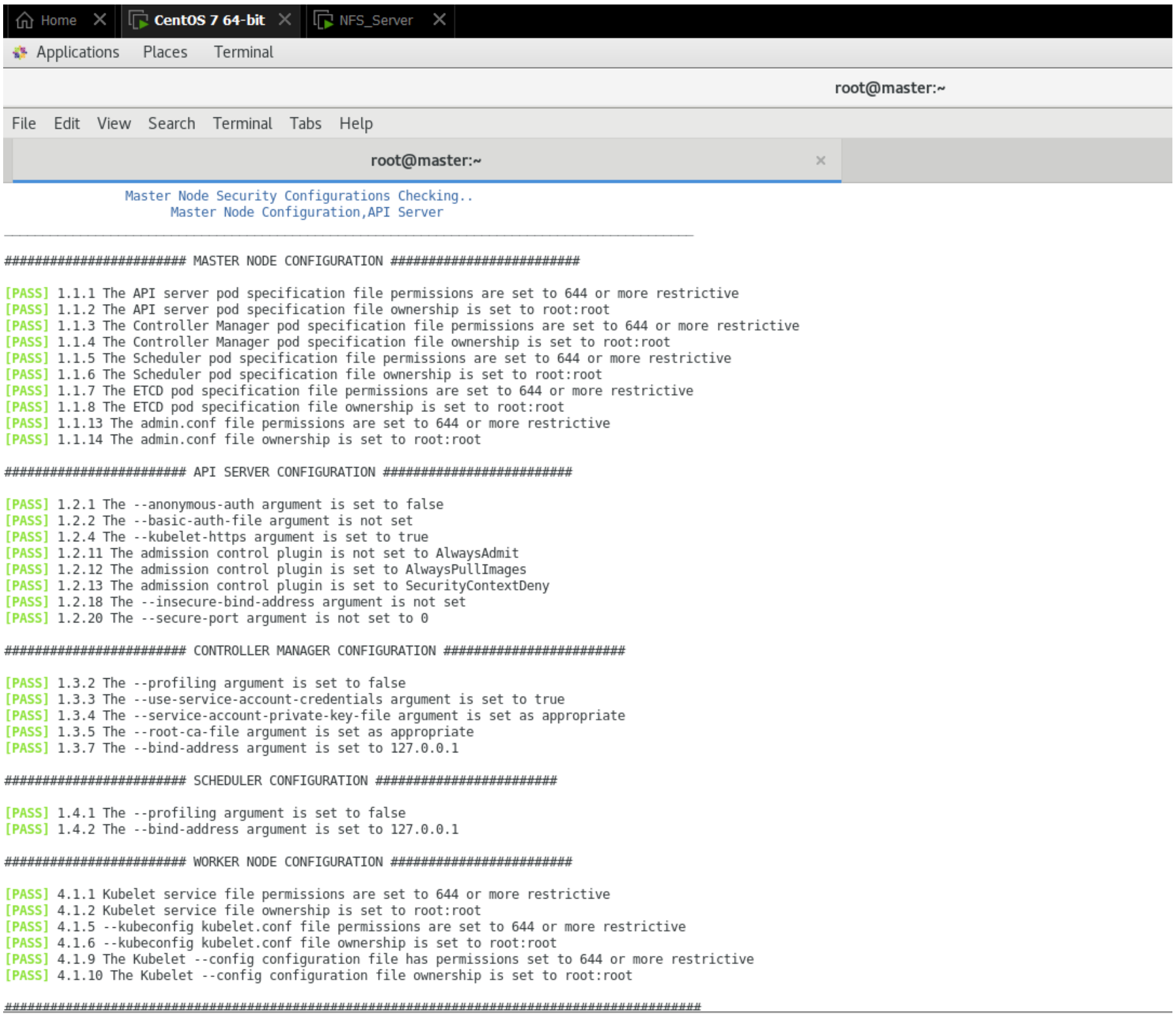
Status of Clustering is checked using below command:

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



```
####### Clustering is DONE!!!! ######
NAMESPACE     NAME                             READY   STATUS    RESTARTS        AGE
kube-system   coredns-78fcd69978-6xkfh         1/1     Running   0               4m18s
kube-system   coredns-78fcd69978-p4qlm         1/1     Running   0               4m18s
kube-system   etcd-master                      1/1     Running   0               4m19s
kube-system   kube-apiserver-master            1/1     Running   0               4m20s
kube-system   kube-controller-manager-master   1/1     Running   0               4m20s
kube-system   kube-proxy-k88fk                 1/1     Running   0               3m2s
kube-system   kube-proxy-xrpt7                 1/1     Running   0               4m18s
kube-system   kube-scheduler-master            1/1     Running   0               4m20s
kube-system   weave-net-2fld2                  2/2     Running   1 (2m5s ago)    3m1s
kube-system   weave-net-xh59c                  2/2     Running   1 (3m10s ago)   3m35s
NAME     STATUS   ROLES                  AGE     VERSION
master   Ready    control-plane,master   4m20s   v1.22.2
worker   Ready    <none>                 3m2s    v1.22.2
```

Step 6 — Scanning and Hardening of Kubernetes



```
                Master Node Security Configurations Checking..
                      Master Node Configuration,API Server
___________________________________________________________________________________________

######################## MASTER NODE CONFIGURATION ########################

[PASS] 1.1.1 The API server pod specification file permissions are set to 644 or more restrictive
[PASS] 1.1.2 The API server pod specification file ownership is set to root:root
[PASS] 1.1.3 The Controller Manager pod specification file permissions are set to 644 or more restrictive
[PASS] 1.1.4 The Controller Manager pod specification file ownership is set to root:root
[PASS] 1.1.5 The Scheduler pod specification file permissions are set to 644 or more restrictive
[PASS] 1.1.6 The Scheduler pod specification file ownership is set to root:root
[PASS] 1.1.7 The ETCD pod specification file permissions are set to 644 or more restrictive
[PASS] 1.1.8 The ETCD pod specification file ownership is set to root:root
[PASS] 1.1.13 The admin.conf file permissions are set to 644 or more restrictive
[PASS] 1.1.14 The admin.conf file ownership is set to root:root

######################## API SERVER CONFIGURATION ########################

[PASS] 1.2.1 The --anonymous-auth argument is set to false
[PASS] 1.2.2 The --basic-auth-file argument is not set
[PASS] 1.2.4 The --kubelet-https argument is set to true
[PASS] 1.2.11 The admission control plugin is not set to AlwaysAdmit
[PASS] 1.2.12 The admission control plugin is set to AlwaysPullImages
[PASS] 1.2.13 The admission control plugin is set to SecurityContextDeny
[PASS] 1.2.18 The --insecure-bind-address argument is not set
[PASS] 1.2.20 The --secure-port argument is not set to 0

######################## CONTROLLER MANAGER CONFIGURATION ########################

[PASS] 1.3.2 The --profiling argument is set to false
[PASS] 1.3.3 The --use-service-account-credentials argument is set to true
[PASS] 1.3.4 The --service-account-private-key-file argument is set as appropriate
[PASS] 1.3.5 The --root-ca-file argument is set as appropriate
[PASS] 1.3.7 The --bind-address argument is set to 127.0.0.1

######################## SCHEDULER CONFIGURATION ########################

[PASS] 1.4.1 The --profiling argument is set to false
[PASS] 1.4.2 The --bind-address argument is set to 127.0.0.1

######################## WORKER NODE CONFIGURATION ########################

[PASS] 4.1.1 Kubelet service file permissions are set to 644 or more restrictive
[PASS] 4.1.2 Kubelet service file ownership is set to root:root
[PASS] 4.1.5 --kubeconfig kubelet.conf file permissions are set to 644 or more restrictive
[PASS] 4.1.6 --kubeconfig kubelet.conf file ownership is set to root:root
[PASS] 4.1.9 The Kubelet --config configuration file has permissions set to 644 or more restrictive
[PASS] 4.1.10 The Kubelet --config configuration file ownership is set to root:root

###########################################################################################
```

5. Bash Shell Script

1. Master Node

```
echo "#### Common Installation script ######"
echo "##### Execute on Manager and worker nodes also ####"
sleep 5
sed -i 's/SELINUX=.*/SELINUX=disabled/g' /etc/selinux/config
setenforce 0
###Enable br_netfilter kernel module
modprobe br_netfilter
echo '1' > /proc/sys/net/bridge/bridge-nf-call-ip6tables
echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
touch /etc/sysctl.d/k8s.conf
echo 'net.bridge.bridge-nf-call-ip6tables = 1' > /etc/sysctl.d/k8s.conf
echo 'net.bridge.bridge-nf-call-iptables = 1' >> /etc/sysctl.d/k8s.conf
###Disable Swap
swapoff -a
###remove swap entry from /etc/fstab
sed -i '/swap/d' /etc/fstab
###Install Docker
yum install -y yum-utils device-mapper-persistent-data lvm2
yum install docker -y

###Add kubernetes repository
cd /etc/yum.repos.d
touch kubernetes.repo
echo '[kubernetes]' > kubernetes.repo
echo 'name=Kubernetes' >> kubernetes.repo
echo 'baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64' >>
kubernetes.repo
echo 'enabled=1' >> kubernetes.repo
echo 'gpgcheck=1' >> kubernetes.repo
echo 'repo_gpgcheck=1' >> kubernetes.repo
echo 'gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg' >> kubernetes.repo
yum install -y kubelet kubeadm kubectl
##Start and enable docker service
systemctl start docker
systemctl enable docker
##start and enable kubelet service
systemctl start kubelet
systemctl enable kubelet
echo "#######Installation completed successfully #####"
echo "Please enter the ip:"
read -r ip
```

```
scp /root/worker-kube.sh $ip:/root/
ssh root@$ip "./worker-kube.sh" > /root/worker-kube.sh
sleep 15s
echo "Adding Firewall Rules"
firewall-cmd --permanent --add-port=6443/tcp
firewall-cmd --permanent --add-port=2379-2380/tcp
firewall-cmd --permanent --add-port=10250/tcp
firewall-cmd --permanent --add-port=10251/tcp
firewall-cmd --permanent --add-port=10252/tcp
firewall-cmd --permanent --add-port=10255/tcp
firewall-cmd --reload
sleep 3s
echo "Creating Token for worker node"
kubeadm init
sleep 10s
touch /root/join-token
chmod u+x /root/join-token
nano /root/join-token
sleep 15s
scp /root/join-token $ip:/root/
sleep 5s
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
export kubever=$(kubectl version | base64 | tr -d '\n')
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$kubever"
ssh root@$ip "./join-token" > /root/join-token
sleep 180s
echo "####### Clustering is DONE!!!! ######"
kubectl get pods --all-namespaces
kubectl get nodes
sleep 5s
echo "####### Scanning and Hardening Start ######"
sh /root/zephyrus.sh
```

2.Worker node Script.

```
echo "#### Common Installation script ######"
echo "##### Execute on Manager and worker nodes also ####"
sleep 5
sed -i 's/SELINUX=.*/SELINUX=disabled/g' /etc/selinux/config
setenforce 0
###Enable br_netfilter kernel module
modprobe br_netfilter
echo '1' > /proc/sys/net/bridge/bridge-nf-call-ip6tables
echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
touch /etc/sysctl.d/k8s.conf
echo 'net.bridge.bridge-nf-call-ip6tables = 1' > /etc/sysctl.d/k8s.conf
echo 'net.bridge.bridge-nf-call-iptables = 1' >> /etc/sysctl.d/k8s.conf
###Disable Swap
swapoff -a
###remove swap entry from /etc/fstab
sed -i '/swap/d' /etc/fstab
###Install Docker
yum install -y yum-utils device-mapper-persistent-data lvm2
yum install docker -y

###Add kubernetes repository
cd /etc/yum.repos.d
touch kubernetes.repo
echo '[kubernetes]' > kubernetes.repo
echo 'name=Kubernetes' >> kubernetes.repo
echo 'baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64'
>> kubernetes.repo
echo 'enabled=1' >> kubernetes.repo
echo 'gpgcheck=1' >> kubernetes.repo
echo 'repo_gpgcheck=1' >> kubernetes.repo
echo 'gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg' >>
kubernetes.repo
yum install -y kubelet kubeadm kubectl
##Start and enable docker service
systemctl start docker
systemctl enable docker
##start and enable kubelet service
systemctl start kubelet
systemctl enable kubelet
echo "#######Installation completed successfully #####"
```

3.Scanning and hardening Script:

```
#!/bin/bash

gr='\033[1;32m'
re='\033[1;31m'
xx='\033[0m'
yw='\033[1;33m'
bl='\033[0;34m'

mstnode1="1.1.1 The API server pod specification file permissions are set to 644 or more restrictive by this script"
mstnode2="1.1.2 The API server pod specification file ownership is set to root:root by this script"
mstnode3="1.1.3 The Controller Manager pod specification file permissions are set to 644 or more restrictive by this script"
mstnode4="1.1.4 The Controller Manager pod specification file ownership is set to root:root by this script"
mstnode5="1.1.5 The Scheduler pod specification file permissions are set to 644 or more restrictive by this script"
mstnode6="1.1.6 The Scheduler pod specification file ownership is set to root:root by this script"
mstnode7="1.1.7 The ETCD pod specification file permissions are set to 644 or more restrictive by this script"
mstnode8="1.1.8 The ETCD pod specification file ownership is set to root:root by this script"
mstnode9="1.1.13 The admin.conf file permissions are set to 644 or more restrictive by this script"
mstnode10="1.1.14 The admin.conf file ownership is set to root:root by this script"

apid1="1.2.1 The --anonymous-auth argument is set to false by this script"
apid2="1.2.2 The --basic-auth-file argument is not set by this script"
apid3="1.2.4 The --kubelet-https argument is set to true by this script"
apid4="1.2.11 The admission control plugin is not set to AlwaysAdmit by this script"
apid5="1.2.12 The admission control plugin is set to AlwaysPullImages by this script"
apid6="1.2.13 The admission control plugin is set to SecurityContextDeny by this script"
apid7="1.2.18 The --insecure-bind-address argument is not set by this script"
apid8="1.2.20 The --secure-port argument is not set to 0 by this script"

cntmgr1="1.3.2 The --profiling argument is set to false by this script"
cntmgr2="1.3.3 The --use-service-account-credentials argument is set to true by this script"
cntmgr3="1.3.4 The --service-account-private-key-file argument is set as appropriate by this script"
cntmgr4="1.3.5 The --root-ca-file argument is set as appropriate by this script"
cntmgr5="1.3.7 The --bind-address argument is set to 127.0.0.1 by this script"

scheduler1="1.4.1 The --profiling argument is set to false by this script"
scheduler2="1.4.2 The --bind-address argument is set to 127.0.0.1 by this script"

workernode1="4.1.1 Kubelet service file permissions are set to 644 or more restrictive by this script"
workernode2="4.1.2 Kubelet service file ownership is set to root:root by this script"
workernode3="4.1.5 --kubeconfig kubelet.conf file permissions are set to 644 or more restrictive by this script"
workernode4="4.1.6 --kubeconfig kubelet.conf file ownership is set to root:root by this script"
workernode5="4.1.9 The Kubelet --config configuration file has permissions set to 644 or more restrictive by this script"
workernode6="4.1.10 The Kubelet --config configuration file ownership is set to root:root by this script"
show(){
printf "${!1}\n"
}

echo ""
echo "  _____           _                            "
echo " |__  /___ _ __ | |__  _   _ _ __ _   _ ___   "
echo "   / // _ \ '_ \| '_ \| | | | '__| | | / __|  "
echo "  / /|  __/ |_) | | | | |_| | |  | |_| \__ \  "
echo " /____\___| .__/|_| |_|\__, |_|   \__,_|___/  "
echo "          |_|          |___/                  "


echo -en '\n'
echo -en
"____________________________________________________________________________________________________"
echo -en '\n'
echo ""
echo -e "${bl}              ~Kubernetes Auditing & Hardening Tool~${xx}"
echo -en '\n'
echo -e "Script made by:"
echo -e "Atish Kukde-210540185011"
echo -e "Ashutosh Sawant-210540185010"
echo -e "Mohsin Attar-210540185026"
echo -e "Kirtiraj Ghuge-210540185018"
echo -e "Pranav Mahajan-210540185035"
echo -en '\n'
echo -e "Scanning and Hardening starting at..." `date`
echo -en
"____________________________________________________________________________________________________"
echo -en '\n'
echo ""
echo -en "${bl}  Master Node Configuration,API Server,Controller Manager,Scheduler &
Worker Node Configuration${xx}"
echo ""
echo -en
"____________________________________________________________________________________________________"
echo ""
echo -en '\n'
echo -e "######################## MASTER NODE CONFIGURATION
#########################"
echo -en '\n'
mstnode1(){
node1=$(stat -c %a /etc/kubernetes/manifests/kube-apiserver.yaml)
if [[ "$node1" == 644 ]];then
        echo -en "${gr}[PASS]${xx}" `show mstnode1`
else
        chmod 644 /etc/kubernetes/manifests/kube-apiserver.yaml
        echo -en "${gr}[PASS]${xx}" `show mstnode1`
fi
}
mstnode1
echo -en '\n'

mstnode2(){
node2=$(stat -c %U:%G /etc/kubernetes/manifests/kube-apiserver.yaml)
if [[ "$node2" == "root:root" ]];then
        echo -en "${gr}[PASS]${xx}" `show mstnode2`
else
        chown root:root /etc/kubernetes/manifests/kube-apiserver.yaml
        echo -en "${gr}[PASS]${xx}" `show mstnode2`
fi
}
mstnode2
echo -en '\n'

mstnode3(){
node3=$(stat -c %a /etc/kubernetes/manifests/kube-controller-manager.yaml)
if [[ "$node3" == 644 ]];then
        echo -en "${gr}[PASS]${xx}" `show mstnode3`
else
        chmod 644 /etc/kubernetes/manifests/kube-controller-manager.yaml
        echo -en "${gr}[PASS]${xx}" `show mstnode3`
fi
}
mstnode3
echo -en '\n'

mstnode4(){
node4=$(stat -c %U:%G /etc/kubernetes/manifests/kube-controller-manager.yaml)
if [[ "$node4" == "root:root" ]];then
        echo -en "${gr}[PASS]${xx}" `show mstnode4`
else
        chown root:root /etc/kubernetes/manifests/kube-controller-manager.yaml
        echo -en "${gr}[PASS]${xx}" `show mstnode4`
fi
}
mstnode4
echo -en '\n'
```

```
mstnode5(){
node5=$(stat -c %a /etc/kubernetes/manifests/kube-scheduler.yaml)
if [[ "$node5" == 644 ]];then
	echo -en "${gr}[PASS]${xx}" `show mstnode5`
else
	chmod 644 /etc/kubernetes/manifests/kube-scheduler.yaml
	echo -en "${gr}[PASS]${xx}" `show mstnode5`
fi
}
mstnode5
echo -en '\n'

mstnode6(){
node6=$(stat -c %U:%G /etc/kubernetes/manifests/kube-scheduler.yaml)
if [[ "$node6" == "root:root" ]];then
	echo -en "${gr}[PASS]${xx}" `show mstnode6`
else
	chown root:root /etc/kubernetes/manifests/kube-scheduler.yaml
	echo -en "${gr}[PASS]${xx}" `show mstnode6`
fi
}
mstnode6
echo -en '\n'


mstnode7(){
node7=$(stat -c %a /etc/kubernetes/manifests/etcd.yaml)
if [[ "$node7" == 644 ]];then
	echo -en "${gr}[PASS]${xx}" `show mstnode7`
else
	chmod 644 /etc/kubernetes/manifests/etcd.yaml
	echo -en "${gr}[PASS]${xx}" `show mstnode7`
fi
}
mstnode7
echo -en '\n'

mstnode8(){
node8=$(stat -c %U:%G /etc/kubernetes/manifests/etcd.yaml)
if [[ "$node8" == "root:root" ]];then
	echo -en "${gr}[PASS]${xx}" `show mstnode8`
else
	chown root:root /etc/kubernetes/manifests/etcd.yaml
	echo -en "${gr}[PASS]${xx}" `show mstnode8`
fi
}
mstnode8
echo -en '\n'
```

```
mstnode9(){
node9=$(stat -c %a /etc/kubernetes/admin.conf)
if [[ "$node9" == 644 ]];then
	echo -en "${gr}[PASS]${xx}" `show mstnode9`
else
	chmod 644 /etc/kubernetes/admin.conf
	echo -en "${gr}[PASS]${xx}" `show mstnode9`
fi
}
mstnode9
echo -en '\n'

mstnode10(){
node10=$(stat -c %U:%G /etc/kubernetes/admin.conf)
if [[ "$node10" == "root:root" ]];then
	echo -en "${gr}[PASS]${xx}" `show mstnode10`
else
	chown root:root /etc/kubernetes/admin.conf
	echo -en "${gr}[PASS]${xx}" `show mstnode10`
fi
}
mstnode10
echo -en '\n'

echo -en '\n'
echo -e "######################## API SERVER CONFIGURATION
########################"
echo -en '\n'

api1(){
anon_auth=$(ps -ef | grep kube-apiserver | grep -o 'anonymous-auth=[^ ,]\+' | awk -F "=" '{print
$2}')

if [ "$anon_auth" == "false" ];then
echo -en "${gr}[PASS]${xx}" `show apid1`
elif [ "$anon_auth" == "" ]
then
echo -en "${gr}[PASS]${xx}" `show apid1`
else
echo -en "${re}[WARN]${xx}" `show apid1`
fi

}

api1
```

```
api2(){
basic_aufi=$(ps -ef | grep kube-apiserver | grep -o 'basic-auth-file=[^ ,]\+' | awk -F "=" '{print $2}')

if [ "$basic_aufi" == "" ];then
echo -en "${gr}[PASS]${xx}" `show apid2`
else
echo -en "${re}[WARN]${xx}" `show apid2`
fi

}

echo ""
api2
echo -en '\n'

api3(){
kube_http=$(ps -ef | grep kube-apiserver | grep -o 'kubelet-https=[^ ,]\+' | awk -F "=" '{print $2}')

if [ "$kube_http" == "" ];then
echo -en "${gr}[PASS]${xx}" `show apid3`
elif [ "$kube_http" == "true"];then
echo -en "${gr}[PASS]${xx}" `show apid3`
else
echo -en "${re}[WARN]${xx}" `show apid3`
fi

}

api3
echo -en '\n'

api4(){
alw_admit=$(ps -ef | grep kube-apiserver | grep -o 'admission-control=[^ ,]\+' | awk -F "=" '{print $2}')

if [[ "$alw_admit" == "" ]];then
echo -en "${gr}[PASS]${xx}" `show apid4`
else
echo -en "${re}[WARN]${xx}" `show apid4`
fi

}

api4
echo -en '\n'
```

```
api5(){
alw_pull=$(ps -ef | grep kube-apiserver | grep -o 'admission-control=[^ ,]\+' | awk -F "=" '{print $2}')
if [[ "$alw_pull" == "AlwaysPullImages" ]];then
echo -en "${gr}[PASS]${xx}" `show apid5`
elif [ "$alw_pull" == "" ];then
echo -en "${gr}[PASS]${xx}" `show apid5`
else
echo -en "${re}[WARN]${xx}" `show apid5`
fi
}
api5
echo -en '\n'
api6(){
sec_cont=$(ps -ef | grep kube-apiserver | grep -o 'admission-control=[^ ,]\+' | awk -F "=" '{print $2}')

if [[ "$sec_cont" == "SecurityContextDeny" ]];then
echo -en "${gr}[PASS]${xx}" `show apid6`
elif [ "$sec_cont" == "" ];then
echo -en "${gr}[PASS]${xx}" `show apid6`
else
echo -en "${re}[WARN]${xx}" `show apid6`
fi
}
api6
echo -en '\n'
api7(){
ins_bind=$(ps -ef | grep kube-apiserver | grep -o 'insecure-bind-address=[^ ,]\+' | awk -F "=" '{print
$2}')

if [ "$ins_bind" == "" ];then
echo -en "${gr}[PASS]${xx}" `show apid7`
else
echo -en "${re}[WARN]${xx}" `show apid7`
fi
}
api7
echo -en '\n'
api8(){
sec_port=$(ps -ef | grep kube-apiserver | grep -o 'secure-port=[^ ,]\+' | awk -F "=" '{print $2}')
if [[ "$sec_port" == "0" ]];then
echo -en "${re}[WARN]${xx}" `show apid8`
elif [ "$sec_port" == "" ];then
echo -en "${gr}[PASS]${xx}" `show apid8`
else
echo -en "${gr}[PASS]${xx}" `show apid8`
fi
}
api8
echo -en '\n'
echo -en '\n'
echo -e "######################## CONTROLLER MANAGER CONFIGURATION
########################"
echo -en '\n'

cntmgr1(){
profiling=$(ps -ef | grep kube-controller-manager | grep -o 'profiling=[^ ,]\+' | awk -F "=" '{print $2}')
if [[ "$profiling" == "false" ]];then
echo -en "${gr}[PASS]${xx}" `show cntmgr1`
else
sed -i '/- --authentication-kubeconfig/i \    \- --profiling=false' /etc/kubernetes/manifests/kube-
controller-manager.yaml
echo -en "${gr}[PASS]${xx}" `show cntmgr1`
fi
}
cntmgr1
echo -en '\n'

cntmgr2(){
uservice=$(ps -ef | grep kube-controller-manager | grep -o 'use-service-account-credentials=[^ ,]\+' |
awk -F "=" '{print $2}')
if [[ "$uservice" == "true" ]];then
        echo -en "${gr}[PASS]${xx}" `show cntmgr2`
else
        sed -i '/- --authentication-kubeconfig/i \    \- --use-service-account-credentials=true'
/etc/kubernetes/manifests/kube-controller-manager.yaml
        echo -en "${gr}[PASS]${xx}" `show cntmgr2`
fi
}
cntmgr2
echo -en '\n'

cntmgr3(){
ackey=$(ps -ef | grep kube-controller-manager | grep -o 'service-account-private-key-file=[^ ,]\+' |
awk -F "=" '{print $2}')
if [[ "$ackey" == "/etc/kubernetes/pki/sa.key" ]];then
        echo -en "${gr}[PASS]${xx}" `show cntmgr3`
else
        sed -i '/- --authentication-kubeconfig/i \    \- --service-account-private-key-
file=/etc/kubernetes/pki/sa.key' /etc/kubernetes/manifests/kube-controller-manager.yaml
        echo -en "${gr}[PASS]${xx}" `show cntmgr3`
fi
}
cntmgr3
echo -en '\n'
```

```
cntmgr4(){
rtca=$(ps -ef | grep kube-controller-manager | grep -o 'root-ca-file=[^ ,]\+' | awk -F "=" '{print $2}')
if [[ "$rtca" == "/etc/kubernetes/pki/ca.crt" ]];then
	echo -en "${gr}[PASS]${xx}" `show cntmgr4`
else
	sed -i '/- --authentication-kubeconfig/i \    \- --root-ca-file=/etc/kubernetes/pki/ca.crt' 
/etc/kubernetes/manifests/kube-controller-manager.yaml
	echo -en "${gr}[PASS]${xx}" `show cntmgr4`
fi
}
cntmgr4
echo -en '\n'

cntmgr5(){
bdaddress=$(ps -ef | grep kube-controller-manager | grep -o 'bind-address=[^ ,]\+' | awk -F "=" '{print 
$2}')
if [[ "$bdaddress" == 127.0.0.1 ]];then
	echo -en "${gr}[PASS]${xx}" `show cntmgr5`
else
	sed -i '/- --authentication-kubeconfig/i \    \- --bind-address=127.0.0.1' 
/etc/kubernetes/manifests/kube-controller-manager.yaml
	echo -en "${gr}[PASS]${xx}" `show cntmgr5`
fi
}
cntmgr5
echo -en '\n'
echo -en '\n'

echo -e "######################## SCHEDULER CONFIGURATION 
########################"
echo -en '\n'

scheduler1(){
scprofile=$(ps -ef | grep kube-scheduler | grep -o 'profiling=[^ ,]\+' | awk -F "=" '{print $2}')
if [[ "$scprofile" == "false" ]];then
	echo -en "${gr}[PASS]${xx}" `show scheduler1`
else
	sed -i '/- --authentication-kubeconfig/i \    \- --profiling=false' /etc/kubernetes/manifests/kube-
scheduler.yaml
	echo -en "${gr}[PASS]${xx}" `show scheduler1`
fi
}
scheduler1
echo -en '\n'
```

```
scheduler2(){
bdaddr=$(ps -ef | grep kube-scheduler | grep -o 'bind-address=[^ ,]\+' | awk -F "=" '{print $2}')
if [[ "$bdaddr" == 127.0.0.1 ]];then
	echo -en "${gr}[PASS]${xx}" `show scheduler2`
else
	sed -i '/- --authentication-kubeconfig/i \    \- --bind-address=127.0.0.1'
/etc/kubernetes/manifests/kube-scheduler.yaml
	echo -en "${gr}[PASS]${xx}" `show scheduler2`
fi
}
scheduler2
echo -en '\n'
echo -en '\n'

echo -e "######################## WORKER NODE CONFIGURATION
########################"
echo -en '\n'

workernode1(){
wrknode1=$(stat -c %a /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf)
if [[ "$wrknode1" == 644 ]];then
	echo -en "${gr}[PASS]${xx}" `show workernode1`
else
	chmod 644 /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
	echo -en "${gr}[PASS]${xx}" `show workernode1`
fi
}
workernode1
echo -en '\n'

workernode2(){
wrknode2=$(stat -c %U:%G /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf)
if [[ "$wrknode2" == "root:root" ]];then
	echo -en "${gr}[PASS]${xx}" `show workernode2`
else
	chown root:root /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
	echo -en "${gr}[PASS]${xx}" `show workernode2`
fi
}
workernode2
echo -en '\n'
```

```
workernode3(){
wrknode3=$(stat -c %a /etc/kubernetes/kubelet.conf)
if [[ "$wrknode3" == 644 ]];then
        echo -en "${gr}[PASS]${xx}" `show workernode3`
else
        chmod 644 /etc/kubernetes/kubelet.conf
        echo -en "${gr}[PASS]${xx}" `show workernode3`
fi
}
workernode3
echo -en '\n'

workernode4(){
wrknode4=$(stat -c %U:%G /etc/kubernetes/kubelet.conf)
if [[ "$wrknode4" == "root:root" ]];then
        echo -en "${gr}[PASS]${xx}" `show workernode4`
else
        chown root:root /etc/kubernetes/kubelet.conf
        echo -en "${gr}[PASS]${xx}" `show workernode4`
fi
}
workernode4
echo -en '\n'

workernode5(){
wrknode5=$(stat -c %a /var/lib/kubelet/config.yaml)
if [[ "$wrknode5" == 644 ]];then
        echo -en "${gr}[PASS]${xx}" `show workernode5`
else
        chmod 644 /var/lib/kubelet/config.yaml
        echo -en "${gr}[PASS]${xx}" `show workernode5`
fi
}
workernode5
echo -en '\n'

workernode6(){
wrknode6=$(stat -c %U:%G /var/lib/kubelet/config.yaml)
if [[ "$wrknode6" == "root:root" ]];then
        echo -en "${gr}[PASS]${xx}" `show workernode6`
else
        chown root:root /var/lib/kubelet/config.yaml
        echo -en "${gr}[PASS]${xx}" `show workernode6`
fi
}
workernode6
echo -en '\n'
echo -en '\n'
```

6. CONCLUSION & FUTURE SCOPE

As stated in the abstract, the project uses several tools to set up Kubernetes cluster between master and worker nodes. Kubernetes cluster is installed on CentOS 7 using Kubeadm and token with the help of bash shell script. We are scanning and hardening the Kubernetes cluster master node, pods, API Server, controller manager, scheduler and worker nodes with the help of CIS Kubernetes Benchmark 2020.

Future Scope:

Even though Kubernetes and CIS benchmark gets updated every now and then, for this purpose we need to update high level security. This can be achieved by this project very efficiently.

7. REFERENCES

1. https://www.docker.com/

2. https://kubernetes.io/

3. https://www.edureka.co/blo

4. **https://github.com/AshuOO7/e-ditiss_project.git**