Cluster Management using Kubernetes

Aakarsh Mavi[1]

1. CDAC-ACTS-PUNE, Maharashtra, India.

Abstract : "Cluster Management using Kubernetes" demonstrates a way to efficiently deploy and manage Web Pages/Web Applications using Kubernetes as the base for deployment.
It works on the principle of using Pods (an inbuilt functionality of Kubernetes). The deployed Pods contain multiple containers and these containers can have multiple Images in it.
Since, Kubernetes is an open source tool it provides you the freedom of letting you move workload anywhere you want. Not only it widely used by Google. It is also quickly being adopted by a lot of big companies and upcoming techies.
For those of you who deploy and manage multi scaled infrastructure of Web Pages you'll appreciate that using Kubernetes for managements will reduce your workload and a lot of automations can be put in place to ease out your lives. Some of the key benefits of using Kubernetes for maintaining are -

- Automates rollouts and rollbacks
- Storage orchestration
- Horizontal scaling
- Self-monitoring

Kubernetes can make use of many more such features. The above points are only to give you a hint of what Kubernetes is capable of.

1. INTRODUCTION

PURPOSE

To Demonstrate the working of Kubernetes and Monitoring the Deployed application, in case our physical storage gets fully utilized then used shared storage for the storing the containers data also monitor the cluster logs using Splunk.

AIMS AND OBJECTIVES:

- To Deploy the the pods of the kubernetes.
- Pods contains the Containers having Images which are pulled from DockerHub, and monitor this pods using Monitoring tool i.e Splunk.
- This Project also aims to provide persistent storage volume for data stored on the cluster

OBJECTIVES

- Deploy the Pods using Kubernetes
- Using Openfiler we used SAN based Shared storage.
- Create a Persistent Volume for the permanent data store.
- Monitor the Containers logs using Splunk tool.

2. SYSTEM REQUIRMENT

- ✓ SOFTWARE REQUIREMENTS:
  - OPENFILER OS
  - VMWARE WORKSTATION
  - UBUNTU OS
  - SPLUNK (TRIAL VERSION)
- ✓ HARDWARE REQUIREMENTS:
  - RAM 16 GB
  - INTEL 4 CORE PROCESSOR
  - NTERNET CONNECTIVITY
- ✓ OS PLATFORM USED:
  - LINUX

3. KUBERNETES

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 for load balancing traffic between different containers and pods.	Docker swarm does auto load balancing of traffic between containers in the cluster.
Rolling Updates & Rollbacks	Can deploy rolling updates and does automatic rollbacks.	Can deploy rolling updates, but not automatic rollback.
DATA Volumes	Can share storage volumes only with the other containers in the same pod.	Can share storage volumes with any other container.
Logging & Monitoring	In-built tools for logging and monitoring.	3rd party tools like ELK stack should be used for logging and monitoring.

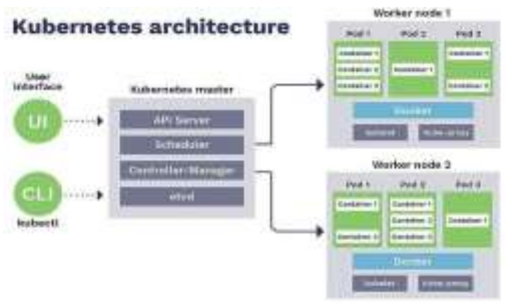
Architecture

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

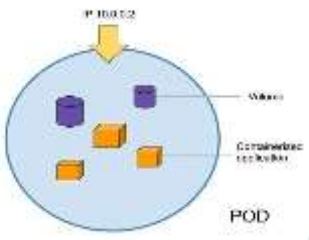


Nodes

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 podsease 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.

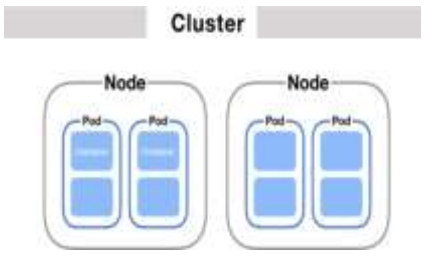
Services

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. Its 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.

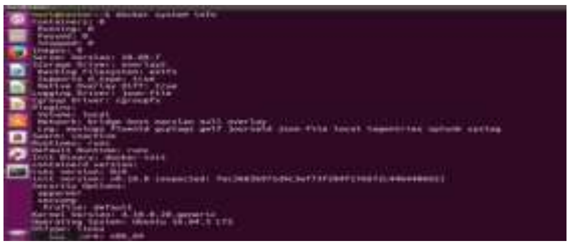
Other than from anPodSpec from the apiserver, there are three ways that a container manifest can be provided to the Kubelet. A Kubelet tracks the state of a pod to ensure that all the containers are running. It provides a heartbeat message every few seconds to the master server. If a replication controller 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

Installation of kubernetes

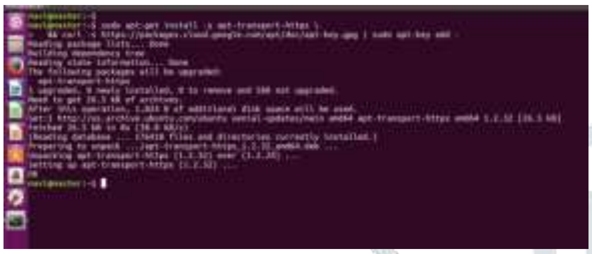
Docker is basically a container engine which uses the Linux Kernel features like namespaces and control groups to create containers on top of an operating system and automates application deployment on the container.



```
navi@master:~$ docker system info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 18.09.7
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version:
runc version: N/A
init version: v0.18.0 (expected: [illegible])
Security Options:
 apparmor
 seccomp
  Profile: default
Kernel Version: 4.15.0.20-generic
Operating System: Ubuntu 16.04.5 LTS
OSType: linux
 [illegible] ure: x86_64
```

Step-1 :- Install apt-transport-https and add kubernetes repository

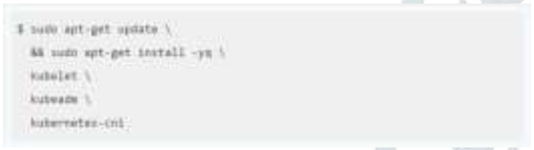
This APT transport allows the use of repositories accessed via the HTTP Secure protocol (HTTPS), also referred to as HTTP over TLS. It is available by default since apt 1.5 and was available before that in the package apt-transport-https. Note that a transport is never called directly by a user but used by APT tools based on user configuration.



```
navi@master:~$
navi@master:~$ sudo apt-get install -y apt-transport-https \
>   && curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be upgraded:
  apt-transport-https
1 upgraded, 0 newly installed, 0 to remove and 190 not upgraded.
Need to get 26.3 kB of archives.
After this operation, 1,024 B of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 apt-transport-https amd64 1.2.32 [26.3 kB]
Fetched 26.3 kB in 0s ([illegible] kB/s)
(Reading database ... 176418 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_1.2.32_amd64.deb ...
Unpacking apt-transport-https (1.2.32) over (1.2.20) ...
Setting up apt-transport-https (1.2.32) ...
OK
navi@master:~$ ▮
```

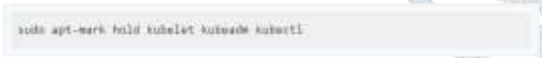
Step-2 :- Install and mark on-hold the following

The kubelet is responsible for scheduling, managing and running containers on your hosts. kubeadm tool is a production-ready convenience utility used to configure the various components that make up a working cluster. The kubernetes-cni package represents the networking components which are not built into Kubernetes directly.



```
$ sudo apt-get update \
  && sudo apt-get install -yq \
  kubelet \
  kubeadm \
  kubernetes-cni
```

We need to put the Kubernetes packages on hold because if we do not, when a newer version is released and we run apt-get upgrade, we could end-up with an unexpected bump in our Kubernetes version.

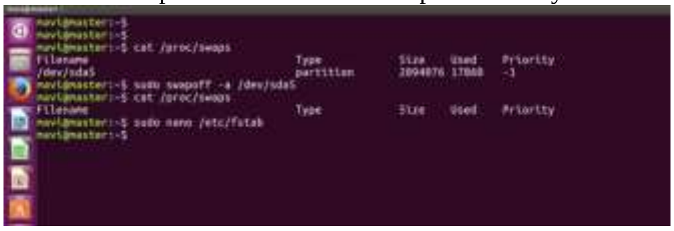


```
sudo apt-mark hold kubelet kubeadm kubectl
```

Step-3 :- Disable Swap Spaces

The Kubernetes maintainers decided that the use of swap memory can lead to unpredictable behaviours and preferred to turn it off, than to contend with any of the side-effects that may come up.

You can check if you have swap enabled by typing in cat /proc/swaps. If you have a swap file or partition enabled then turn it off with swapoff. You can make this permanent by commenting out the swap file in /etc/fstab.



```
navi@master:~$
navi@master:~$
navi@master:~$ cat /proc/swaps
Filename                                Type            Size    Used    Priority
/dev/sda5                               partition       2094076 17868   -1
navi@master:~$ sudo swapoff -a /dev/sda5
navi@master:~$ cat /proc/swaps
Filename                                Type            Size    Used    Priority
navi@master:~$ sudo nano /etc/fstab
navi@master:~$
```

<u>At this point you should have run the above steps on each of your nodes. These are all common steps.</u>

Step-4 :- Cluster Initialization

Initialize your cluster with **kubeadm**

You only need to run this step on your designated **master node**, for us that was **master.**

```
[illegible]
```

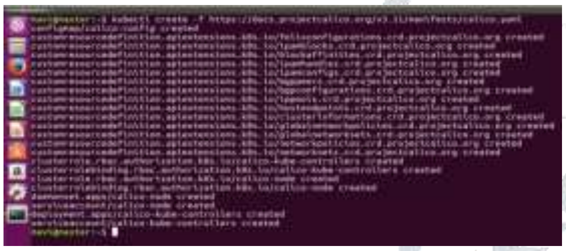
Copy kube config

To be able to use **kubectl** command to connect and interact with the cluster, the user needs kube config file.
In my case, the user account is **mavi**

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

Deploy Calico network

This has to be done as the user in the above step (in our case it is **mavi**)



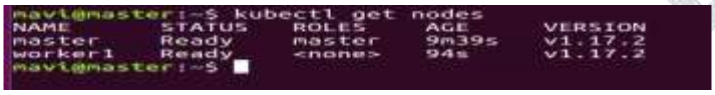
```
[illegible]
```

Use command **Kubectl get nodes** to check number of pods that are currently in the cluster.

Now after the network command use the Cluster Join command that we got after the Kube initiation command on **Worker nodes**.

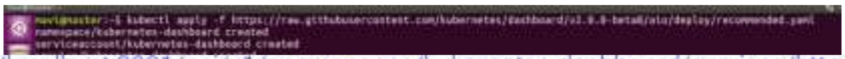
```
[illegible]
```

After using the join command on Worker node the output we get on master node for number of nodes connected is :-



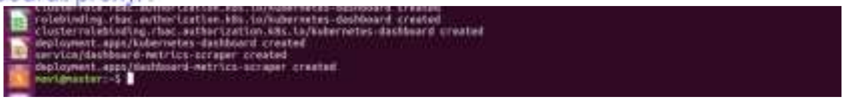
```
mavi@master:~$ kubectl get nodes
NAME      STATUS   ROLES    AGE     VERSION
master    Ready    master   9m39s   v1.17.2
worker1   Ready    <none>   94s     v1.17.2
mavi@master:~$ 
```

Now both master and worker1 are in the cluster.
But still the management of the cluster can only be done through **Command Line** only so we need to generate the **UI Dashboard for Kubernetes.**
Using the following command the UI Dashboard is generated.



```
mavi@master:~$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta8/aio/deploy/recommended.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
```

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/.

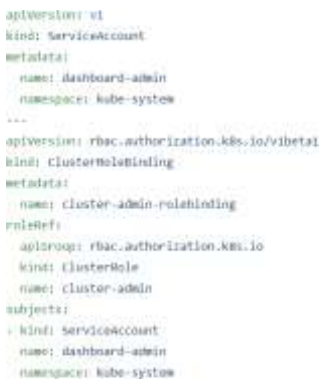


```
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
mavi@master:~$ 
```

Now if we use **Kubectl pxoxy** on command line and check the browser on the following URL

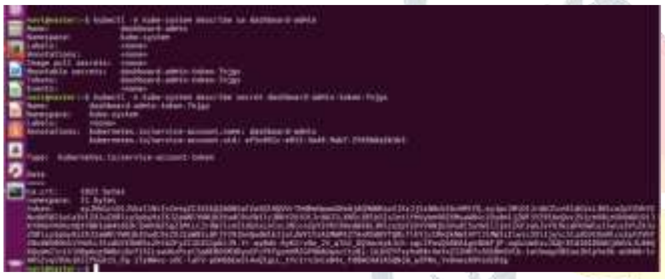
The Dashboard login below is available now.

But if we login through the token, we would still not be able to get information of the cluster the default user for Dashboard does not cluster admin privileges so we need to create another user with privileges.



```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dashboard-admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: dashboard-admin
  namespace: kube-system
```

A user by the name **dashboard-admin** is created with **cluster admin** privileges.
Now, to login into the UI we need the secret **token** is generated for this **dashboard-user.** This can be retrieved using the following commands.



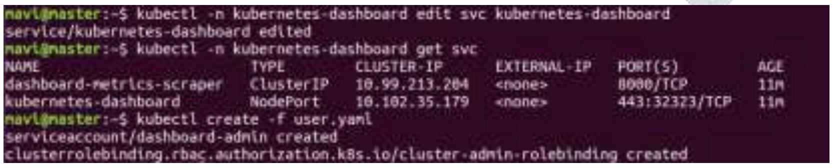
```
navi@master:~$ kubectl -n kube-system describe sa dashboard-admin
Name:                dashboard-admin
Namespace:           kube-system
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   dashboard-admin-token-7rjgn
Tokens:              dashboard-admin-token-7rjgn
Events:              <none>
navi@master:~$ kubectl -n kube-system describe secret dashboard-admin-token-7rjgn
Name:         dashboard-admin-token-7rjgn
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: dashboard-admin
              kubernetes.io/service-account.uid: [illegible]

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     [illegible] bytes
namespace:  11 bytes
token:      [illegible]
navi@master:~$
```

As we discussed earlier that now we need to access the UI by the link

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/.

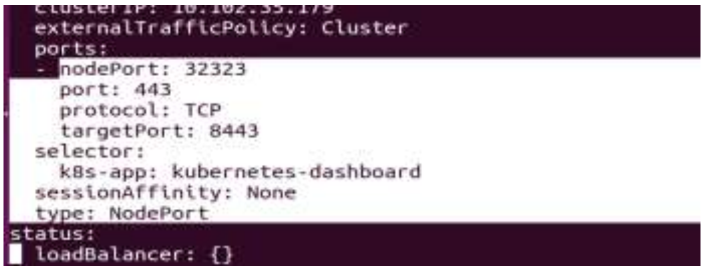
But it is very difficult to remember such a link so, to overcome this issue the dashboard is exposed as a service and the **NodePort** is fixed so that it can easily be accessed.

The following changes are made by using the command:-



```
navi@master:~$ kubectl -n kubernetes-dashboard edit svc kubernetes-dashboard
service/kubernetes-dashboard edited
navi@master:~$ kubectl -n kubernetes-dashboard get svc
NAME                        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)         AGE
dashboard-metrics-scraper   ClusterIP   10.99.213.204   <none>        8000/TCP        11m
kubernetes-dashboard        NodePort    10.102.35.179   <none>        443:32323/TCP   11m
navi@master:~$ kubectl create -f user.yaml
serviceaccount/dashboard-admin created
clusterrolebinding.rbac.authorization.k8s.io/cluster-admin-rolebinding created
```

The changes made are for NodePort only.



```
  clusterIP: 10.102.35.179
  externalTrafficPolicy: Cluster
  ports:
  - nodePort: 32323
    port: 443
    protocol: TCP
    targetPort: 8443
  selector:
    k8s-app: kubernetes-dashboard
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

Now the **dashboard service** can be accessed on port **32323** of **Node worker1** in our case.

At this stage we are done with creating a cluster with two nodes namely master and worker1 also we have generated the UI for Kubernetes along with a privileged user for accessing the dashboard.

Now we will move forward with attach a SAN (Storage Area Network) e.g. Openfiler based storage to the master node so that we can use it for combined storge.

4. OPENFILER

Openfiler is an Linux based operating system that provides file-based network-attached storage (NAS) and block-based storage area network(SAN) or we can say that Openfiler provides a simple way to deploy and manage networked storage. Openfiler helps you to build the very powerful & reliable networked storage solution which is easily managed by the Openfiler browser based management UI. It provides the file based storage networking protocols such as NFS & CIFS which ensures cross platform compatibility for windows, Linux & UNIX. It also provides the iSCSI Target & Fiber channel features which can be used in the virtualized environments such as VMware.

Openfiler helps the Storage administrators & IT Team to deploy, manage & utilize the storage efficiently within multi-platform network. It provides the range of the storage networking protocol which makes it excellent choice to use it. It also helps companies to save the cost of storage system because it is an open source storage solution. Openfiler can be directly deployed on the Physical Hardware or also can be used in the Virtualized Environment.

Key Benefits of Openfiler:

- ✓ RAID 0,1,5,6,10 support
- ✓ Bare metal or Virtualization Installation
- ✓ Unified storage SAN & NAS
- ✓ CIFS, NFS, FTP, HTTP protocol support
- ✓ Intuitive Web Based UI & Management
- ✓ Access Control List

VMware System Requirement for Installation of Openfiler:

- ✓ VMware Compatible ESX, VMware Player
- ✓ Symbios or Buslogic virtual SCSI disk driver
- ✓ IDE virtual disk driver
- ✓ 64 bit Hypervisor
- ✓ 2GB or more Memory
- ✓ Virtual NIC

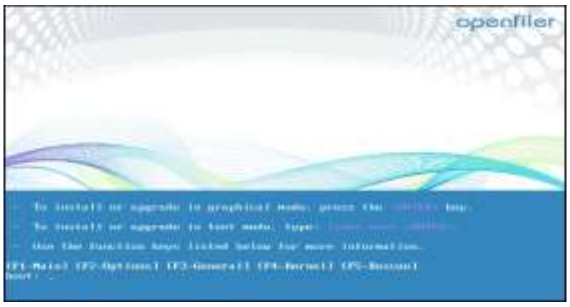
Step by Step Installation of Openfiler:

Step 1: Download the ISO from Openfiler Website.

- ✓ Mount the ISO to your Server
- ✓ Start the Server

Step 2: This is the first Installation screen you will see.

To install it in Graphical Mode press Enter or you can also install it using text mode by second option.

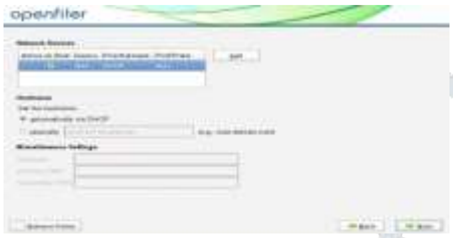


```
  - To install or upgrade in graphical mode, press the <ENTER> key.
  - To install or upgrade in text mode, type: linux text <ENTER>.
  - Use the function keys listed below for more information.
[F1-Main] [F2-Options] [F3-General] [F4-Kernel] [F5-Rescue]
boot: _
```

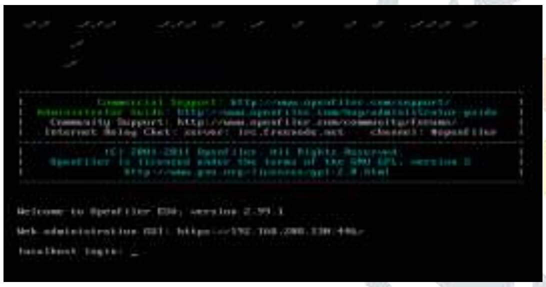
Step 3: Choose the appropriate Language & click Next to proceed.

Step 4: Now we will specify the partition here.
But we are going with the default option "Remove all partitions on selected drives & create default layout."

Step 5: Now we will configure the network for Openfiler.
We setup a "automatically via DHCP" i.e we use DHCP network for the Network Configuration.



Step 8: Now select the Time zone (select city as Asia/Calcutta)for your server and next proceed.
Step 9: Now set the Root user Password
Reboot the System after Installation then we get Openfiler CLI window that contain IP address and port for web UI.



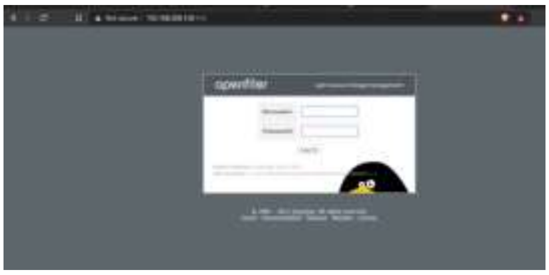
```
 -------------------------------------------------------------------------
 |       Commercial Support: http://www.openfiler.com/support/            |
 |  Administrator Guide: http://www.openfiler.com/buy/administrator-guide |
 |   Community Support: http://www.openfiler.com/community/forums/        |
 |  Internet Relay Chat: server: irc.freenode.net   channel: #openfiler   |
 -------------------------------------------------------------------------
 |           (C) 2001-2011 Openfiler. All Rights Reserved.                |
 |   Openfiler is licensed under the terms of the GNU GPL, version 2      |
 |            http://www.gnu.org/licenses/gpl-2.0.html                    |
 -------------------------------------------------------------------------

Welcome to Openfiler ESA, version 2.99.1

Web administration GUI: https://192.168.208.130:446/

localhost login: _
```

Step 10: Go to the Browser and write https://192.168.208.130:446 in URL.

Volume Creation

Now, we use iSCSI protocol for the shared storage, to configure iSCSI storage in Openfiler go through following steps:

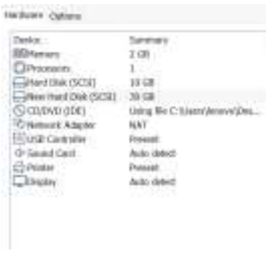
Step 1: Login to Openfiler using web browser.



The default username and password of the Openfiler is:
Username: openfiler
Password: password
If you want you can change this password.

Now after login you get the following UI page, which consist of Openfiler system IP, Hostname, Kernel version etc. following information.

Step 2 : Now add the one more Hard disk to the openfiler.



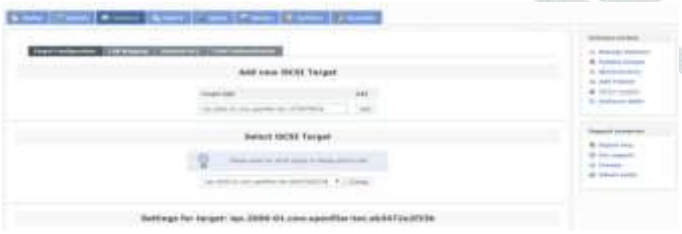
Hardware Options

Device	Summary
Memory	2 GB
Processors	1
Hard Disk (SCSI)	10 GB
New Hard Disk (SCSI)	20 GB
CD/DVD (IDE)	Using file C:\Users\[illegible]\Des...
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Now check in UI there will be 2 disk drive shown in Block device management.

Step 3: Create partition and add the volume using volume group. At this stage we have successfully created iSCSI partition.

Step 4: At this stage we have successfully created volume group & volume. Now we add the iSCSI Target & LUN mapping.

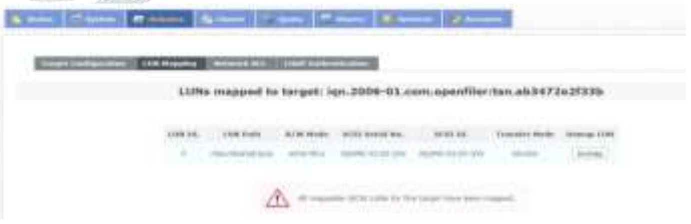


Target Configuration | LUN Mapping | Network ACL | CHAP Authentication

Add new iSCSI Target

Select iSCSI Target

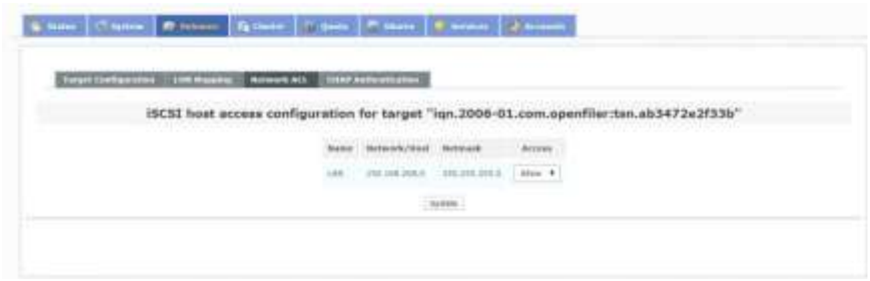
Settings for target: iqn.2006-01.com.openfiler:tsn.ab3472e2f33b



Target Configuration | LUN Mapping | Network ACL | CHAP Authentication

LUNs mapped to target: iqn.2006-01.com.openfiler:tsn.ab3472e2f33b

Step 5: Now we allow the Network Access to iSCSI Target so we choose the Allow option.



Target Configuration | LUN Mapping | Network ACL | CHAP Authentication

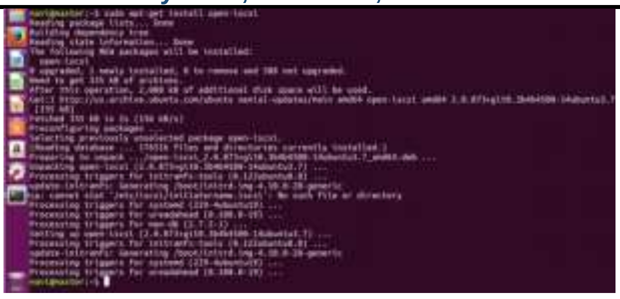
iSCSI host access configuration for target "iqn.2006-01.com.openfiler:tsn.ab3472e2f33b"

Name	Network/Host	Netmask	Access
[illegible]	[illegible]	[illegible]	Allow

Update

Step 6: Now come to Master node where we want to share a storage. And install open-iscsi packages.

```
sudo apt-get install open-iscsi
```



```
[illegible]:~$ sudo apt-get install open-iscsi
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  open-iscsi
[illegible]
```

Steps 7: After that open the configuration file and change the username, password and other parameter.

node.startup=automatic node.session.auth.username=MY-ISCSI-USER node.session.auth.password=MY-ISCSI-PASSWORD discovery.sendtargets.auth.username=MY-ISCSI-USER

```
discovery.sendtargets.auth.password=MY-ISCSI-PASSWORD
node.session.timeo.replacement_timeout=120
node.conn[0].timeo.login_timeout=15
node.conn[0].timeo.logout_timeout=15
node.conn[0].timeo.noop_out_interval=10
node.conn[0].timeo.noop_out_timeout=15 node.session.iscsi.InitialR2T =
No
node.session.iscsi.ImmediateData=Yes
node.session.iscsi.FirstBurstLength=262144
node.session.iscsi.MaxBurstLength=16776192
node.conn[0].iscsi.MaxRecvDataSegmentLength = 65536
```

Save and close the file and Restart the open-iscsi service

Step 8: Now we run a discovery against the iscsi target host.

sudo iscsiadm -m discovery -t sendtarget -p 192.168.208.130

Now this command generates a record id found by the discovery. This record id is used for login purpose.

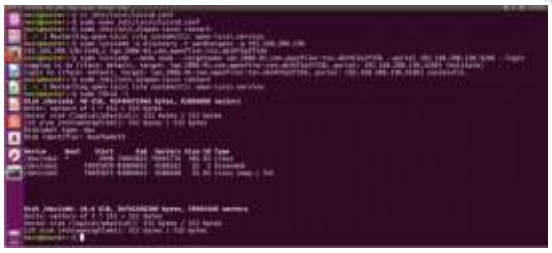
sudo iscsiadm –mode node –targetname <record id> --portal 192.168.208.130:3260 –login

Again restart the service

sudo /etc/init.d/open-iscsi restart

Step 9: check the connected disk to the system using

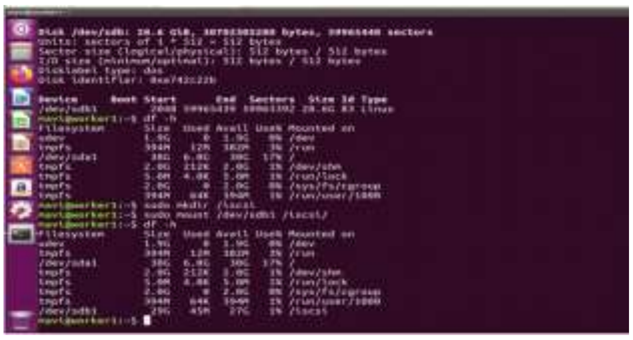
sudo fdisk -l



```
[illegible]
```

Here you can see ***/dev/sdb*** is added .

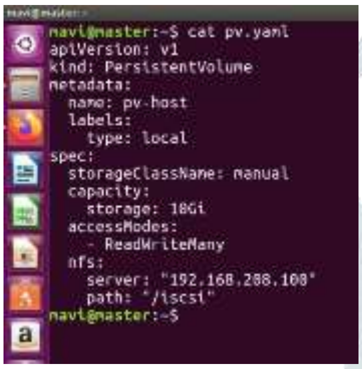
Step 10: Create a partition using fdisk /*dev/sdb* and create a 30 GB primary partition .
Now format the partition in **ext3**. And create a directory in **/** named as **/iscsi**
Using mount command we **mount /*dev/sdb1 to /iscsi* dir**
Now check the disk usage using **df –h**



```
Disk /dev/sdb: 28.6 GiB, 30702302208 bytes, 59965440 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xe742c22b

Device     Boot Start      End  Sectors  Size Id Type
/dev/sdb1        2048 59965439 59963392 28.6G 83 Linux
navi@worker1:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            1.9G     0  1.9G   0% /dev
tmpfs           394M   12M  382M   3% /run
/dev/sda1        38G  6.0G   30G  17% /
tmpfs           2.0G  212K  2.0G   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           2.0G     0  2.0G   0% /sys/fs/cgroup
tmpfs           394M   64K  394M   1% /run/user/1000
navi@worker1:~$ sudo mkdir /iscsi
navi@worker1:~$ sudo mount /dev/sdb1 /iscsi/
navi@worker1:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            1.9G     0  1.9G   0% /dev
tmpfs           394M   12M  382M   3% /run
/dev/sda1        38G  6.0G   30G  17% /
tmpfs           2.0G  212K  2.0G   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           2.0G     0  2.0G   0% /sys/fs/cgroup
tmpfs           394M   64K  394M   1% /run/user/1000
/dev/sdb1        29G   45M   27G   1% /iscsi
navi@worker1:~$
```

As we have attached the Openfiler to the master node, now we will use this storage to create persistent storage that will be further attached to nodes for combined storage

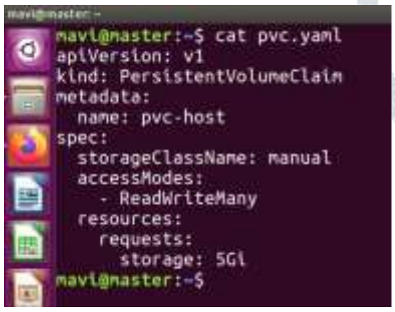
YAML file to create the Persistent Volume



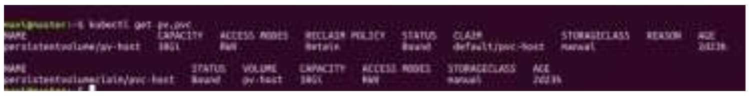
```
navi@master:~$ cat pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-host
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: "192.168.208.100"
    path: "/iscsi"
navi@master:~$
```

The Persistent Volume created has to be claimed for further usage.

YAML file to create the Persistent Volume Claim



```
navi@master:~$ cat pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-host
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
navi@master:~$
```



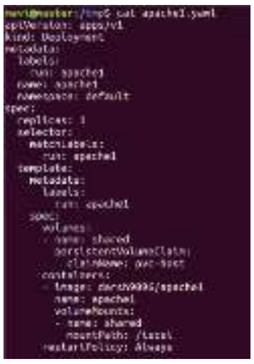
```
navi@master:~$ kubectl get pv,pvc
NAME                      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
persistentvolume/pv-host  10Gi       RWX            Retain           Bound    default/pvc-host   manual                  2d23h

NAME                            STATUS   VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/pvc-host  Bound    pv-host   10Gi       RWX            manual         2d23h
navi@master:~$
```

Now the setup for combined storage is ready for usage.

Deployment of Pods in the cluster using the combined storage.

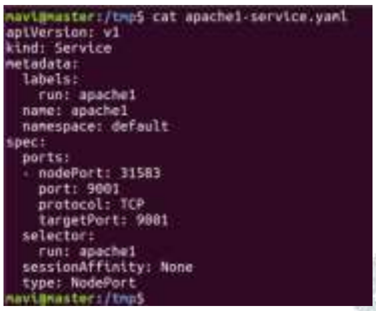
We will use a Ubuntu image for deployment on which http service is running on port 80 and has been exposed on port 9001 and is saved in our personal DockerHub repository.

Now deploying the image using YAML file and integrating the Persistent Volume into it.



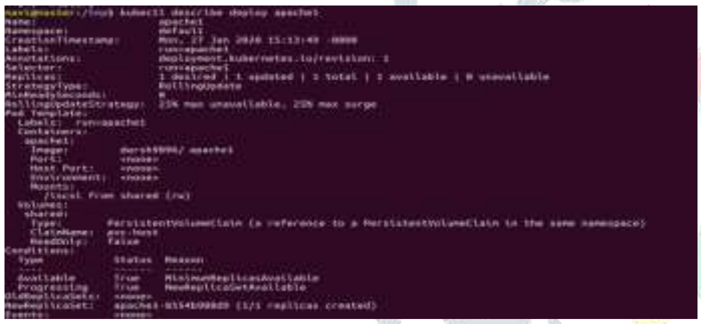
```
navi@master:/tmp$ cat apache1.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: apache1
  name: apache1
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      run: apache1
  template:
    metadata:
      labels:
        run: apache1
    spec:
      volumes:
      - name: shared
        persistentVolumeClaim:
          claimName: pvc-test
      containers:
      - image: darsh9996/apache1
        name: apache1
        volumeMounts:
        - name: shared
          mountPath: /local
      restartPolicy: Always
```

For the ease of access we will create a service for this deployment.



```
navi@master:/tmp$ cat apache1-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    run: apache1
  name: apache1
  namespace: default
spec:
  ports:
  - nodePort: 31583
    port: 9001
    protocol: TCP
    targetPort: 9001
  selector:
    run: apache1
  sessionAffinity: None
  type: NodePort
navi@master:/tmp$
```

Checking the Newly deployed image using command **kubectl describe deployment apache1**



```
navi@master:/tmp$ kubectl describe deploy apache1
Name:                   apache1
Namespace:              default
CreationTimestamp:      Mon, 27 Jan 2020 15:13:49 -0000
Labels:                 run=apache1
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               run=apache1
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=apache1
  Containers:
   apache1:
    Image:        darsh9996/ apache1
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:
      /local from shared (rw)
  Volumes:
   shared:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  pvc-host
    ReadOnly:   false
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   apache1-6554b99889 (1/1 replicas created)
Events:          <none>
```

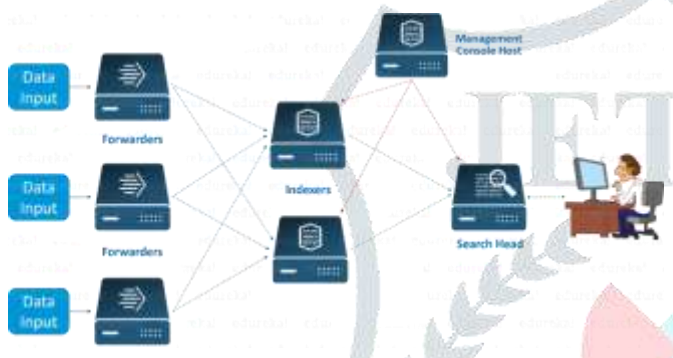
Now check for the deployment on the browser using the NoPort number 31583 and also on the UI interface.

Also the both apache1 and apche2 deployment used the diff images and are running on different Node Ports.

5. SPLUNK

Splunk is a software technology which is used for monitoring, searching, analyzing and visualizing the machine generated data in real time. It can monitor and read different type of log files and stores data as events in indexers. This tool allows you to visualize data in various forms of dashboards.

Splunk Architecture:



Splunk Architecture Diagram

Componentes Of Splunk Architecture:

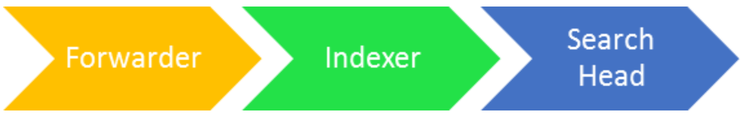
Fundamental components of Splunk architecture:

Universal Forward (UF):
Universal forward or UF is a lightweight component which pushes the data to the heavy Splunk forwarder. You can install Universal Forward at client side or application server. The job of this component is only to forward the log data.

Indexer :
Indexer helps you to store and index the data. It improves Splunk search performance. By default, Splunk automatically performs the indexing. For example, host, source, and date & time.

Search head (SH):
Search head is used to gain intelligence and perform reporting.

How Splunk Works?



Forwarder:
Forwarder collect the data from remote machines then forwards data to the Index in real-time

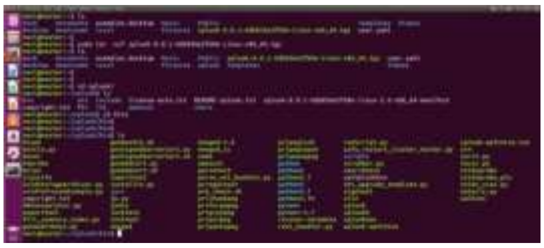
Indexer:
Indexer process the incoming data in real-time. It also stores & Indexes the data on disk.

Search Head:

End users interact with Splunk through Search Head. It allows users to do search, analysis & Visualization.

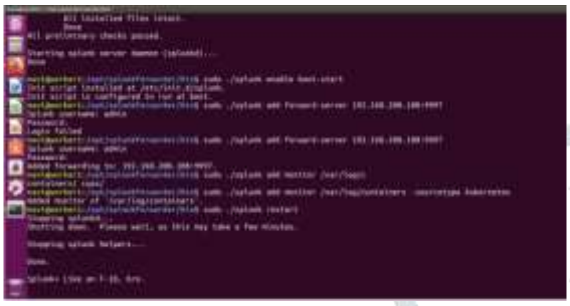
Step by step Installtion:

Step 1 : Installation of splunk Indexer on Master Node



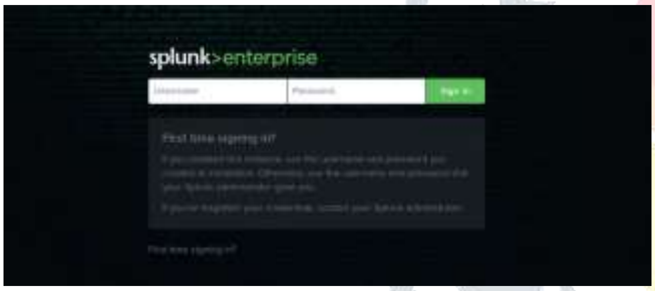
```
[illegible]
```

Step 2: After that we install Splunk Forwarder On worker Node and monitor a kubernetes logs there path is /var/log/containers and source type is 'kubernetes'

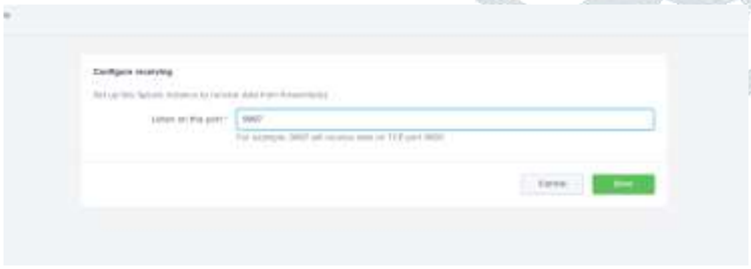


```
        All installed files intact.
        Done
All preliminary checks passed.

Starting splunk server daemon (splunkd)...
Done

root@worker1:/opt/splunkforwarder/bin$ sudo ./splunk enable boot-start
Init script installed at /etc/init.d/splunk.
Init script is configured to run at boot.
root@worker1:/opt/splunkforwarder/bin$ sudo ./splunk add forward-server [illegible]:9997
Splunk username: admin
Password:
Login failed
root@worker1:/opt/splunkforwarder/bin$ sudo ./splunk add forward-server [illegible]:9997
Splunk username: admin
Password:
Added forwarding to: [illegible]:9997.
root@worker1:/opt/splunkforwarder/bin$ sudo ./splunk add monitor /var/logs
[illegible]
root@worker1:/opt/splunkforwarder/bin$ sudo ./splunk add monitor /var/log/containers -sourcetype kubernetes
Added monitor of '/var/log/containers'.
root@worker1:/opt/splunkforwarder/bin$ sudo ./splunk restart
Stopping splunkd...
Shutting down.  Please wait, as this may take a few minutes.

Stopping splunk helpers...

Done.

[illegible]
```

Step 3: go to the web browser and type master:8000



Step 4: we set the port 9997 to receive a data from splunk forwarder



Configure receiving

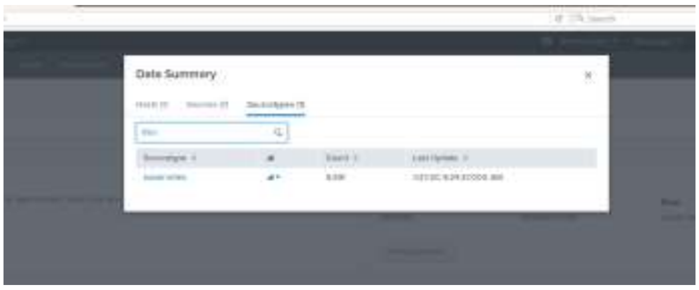
Set up this Splunk instance to receive data from forwarders.

Listen on this port * 9997

For example, 9997 will receive data on TCP port 9997.

Step 5: In the data summary we select kubernetes to see the kubernetes logs

Step 6 : Completion of splunk forwarder configuration in the remote host(client) and connection to master:



Step 7: After we get kubernetes containers logs.

Features of Splunk:

- ***Flexible Data Input***

  Collect and index log data from any source imaginable from network traffic to web servers to custom applications.

- ***Search and Investigate Across All Logs***

  With Splunk Light you have one centralized place to search and find the source of the fire.

- ***Real-Time Search***

  Search real-time streaming data and indexed historical data from the same interface. User can analyze current behavior and activity and see the historical context to get the full picture.

- ***Monitor and Alert Proactively***

  Use your centralized log data to become more proactive. Rather than simply reacting to ad hoc incidents or problems, Splunk Light provides active monitoring and alerting.

IMPLICATIONS FOR FUTURE RESEARCH

The Project has been developed meeting all the requirements thoroughly.
It has been a substantial stride in the phase of managing clusters of kubernetes pods and deploying Web Applications in them. Yet there remain some aspects left unexplored with regard to taking the development of the project to another level.
The following can be significant future implications:

a) Making User experience enriched with availability web applications at all times and is also portable.
b) Controlled and automated deployments as well as updates.
c) More money can be saved by optimizing infrastructural resources thanks to the more efficient use of hardware
d) Containers can be orchestrated on multiple hosts
e) Resources and applications can be scaled in real time

The capabilities of our project by providing more number of nodes on the top of Type-1 Hypervisors and try to increase the scalability of the project with the use of the Kubernetes. Apart from this we can use different tools like Selenium for testing our application and for monitoring also we can use ELK for log monitoring. We can use RAID 1 for mirroring the disk in iSCSI so that we can restore data in case of failure of hard drive.

In the future considering, the above-mentioned areas, the project can be carried further with its development and can contribute meaningfully to serve the organization with the ease of deploying applications.

REFERENCES

1) https://www.smartsheet.com/devops-tools
2) https://alanxelsys.com/citrixxenserver-quick-start/
3) https://www.youtube.com/results?search_query=xen+hypervisor+t utorl
4) https://www.youtube.com/watch?v=p7-U1_E_j3w
5) https://kubernetes.io/
6) https://geek-university.com/splunk/splunk-home/
7) https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml
8) https://www.cyberciti.biz/faq/howto-setup-debian-ubuntu-linux- iscsi-initiator/