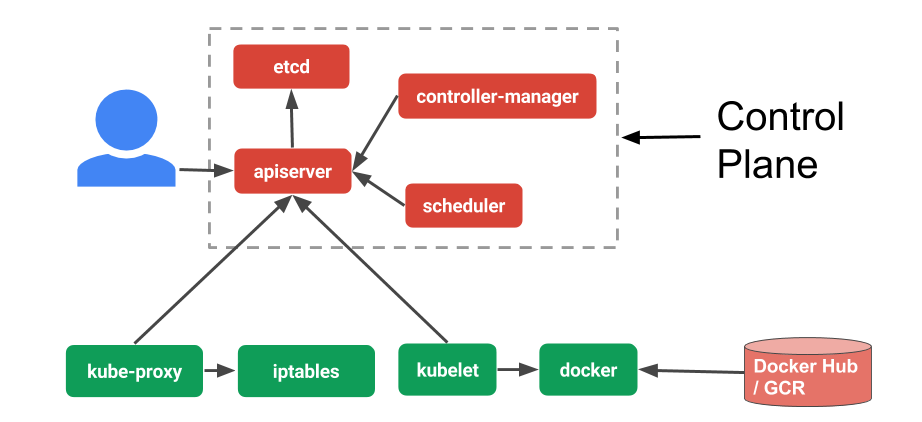
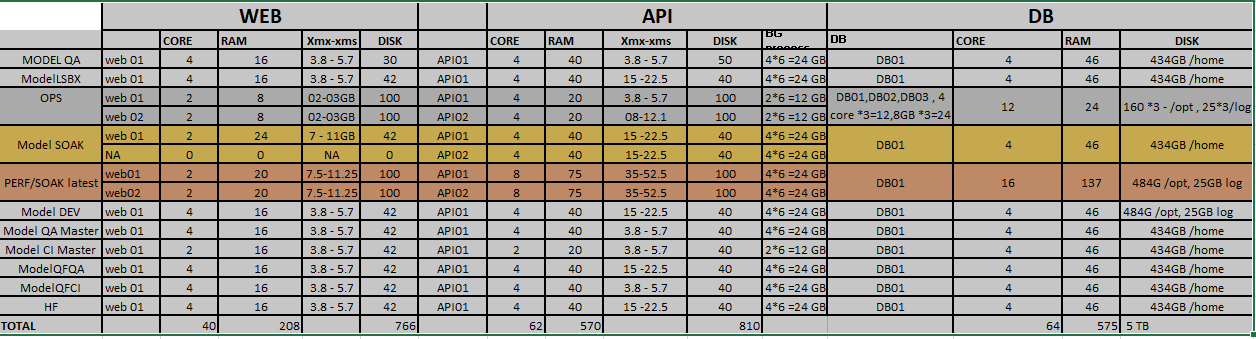
**MC Kubernetes DEV Environment Setup**

# Kubernetes Kubeadm Architecture.



# Existing Dev Environment Setup



# Plan:

**Steps to build environment:**

* Build DB VM ‘s for each Environment.
* Create Name space per environment type (ex: Perf,ops,Lsbx.QA)
* Create NFS storage class and create persistent volume for both WEB and API specific to namespace.
* Perform Deployment and Add expose application using service type node port/Ingress controller specific to name space.
* Add Horizontal pod Autoscaler to auto scale instances.
* Create a Kubernetes Dashboard.
* Enable logging and monitoring (Logic Monitor and sumologic)

|  |  |  |
| --- | --- | --- |
| **Environment** | **WEB(2 container)** | **API(2 Container)** |
| Perf | 2 core/10 GB each | 4 core /38 GB each |
| OPS | 2 core/4GB each | 2 core /10 GB each |
| LSBX | 2 core/8 GB each | 2 core /20 GB each |
| QA | 2 core/8 GB each | 2 core /20 GB each |
|  |  |  |

**Install Kubernetes Cluster using kubeadm**

Documentation to set up a Kubernetes cluster on **CentOS 7** Virtual machines. Setting up cluster with one master node and two worker nodes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Role | Host | Ip | OS | RAM | CPU |
| Master/worker | HYD-DEVOPSMC01.HAICPM.COM | 192.168.136.16 | CentOS 7.3.1611 | 180 GB | 20 |
| Worker | HYD-DEVOPSMC02.HAICPM.COM | 192.168.136.20 | CentOS 7.3.1611 | 180 GB | 20 |
| Worker | HYD-DEVOPSMC03.HAICPM.COM | 192.168.136.47 | CentOS 7.3.1611 | 180 GB | 20 |
|  |  |  |  |  |  |

**On both master and worker**

### **Pre-requisites**

##### **Install, enable and start docker service**

##### **Disable SELinux**

|  |
| --- |
| setenforce 0  sed -i --follow-symlinks 's/^SELINUX=enforcing/SELINUX=disabled/' /etc/sysconfig/selinux |

##### **Disable Firewall**

|  |
| --- |
| systemctl disable firewalld  systemctl stop firewalld |

##### **Disable swap**

|  |
| --- |
| sed -i '/swap/d' /etc/fstab  swapoff -a |

##### Update sysctl settings for Kubernetes networking

|  |
| --- |
| cat >>/etc/sysctl.d/kubernetes.conf<<EOF  net.bridge.bridge-nf-call-ip6tables = 1  net.bridge.bridge-nf-call-iptables = 1  EOF  sysctl --system |

### **Kubernetes Setup**

### **Add yum repository**

|  |
| --- |
| cat >>/etc/yum.repos.d/kubernetes.repo<<EOF  [kubernetes]  name=Kubernetes  baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64  enabled=1  gpgcheck=1  repo\_gpgcheck=1  gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg  https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg  EOF |

##### **Install Kubernetes**

|  |
| --- |
| yum install -y kubeadm kubelet Kubectl  yum -y install nfs-common cifs-utils nfs4-acl-tools nfs-utils ( check dynamic NFS provisionar) |

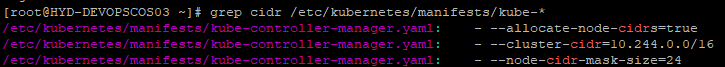
##### **Enable and Start kubelet service**

|  |
| --- |
| systemctl enable kubelet  systemctl start kubelet |

## On master

##### **Initialize Kubernetes Cluster**

|  |
| --- |
| kubeadm init --apiserver-advertise-address= 192.168.136.16 --pod-network-cidr=10.244.0.0/16 |



**Create new user**

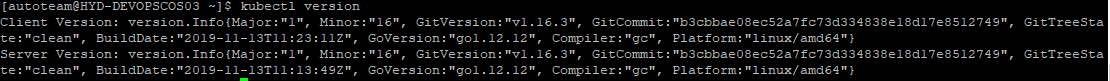
|  |
| --- |
| useradd -d /home/admin admin  usermod -aG docker admin  passwd admin  or  useradd -d /opt/autoteam autoteam  usermod -aG docker autoteam  passwd autoteam |

Make autoteam /admin as Sudo user:

|  |
| --- |
| cat /etc/sudoers | grep admin  %admin ALL=(ALL:ALL) ALL  (or)  cat /etc/sudoers | grep autoteam  %autoteam ALL=(ALL:ALL) ALL |

To be able to use kubectl command to connect and interact with the cluster, the user needs kube config file.

|  |
| --- |
| mkdir /home/admin/.kube  cp -r /etc/kubernetes/admin.conf /home/admin/.kube/config  chown -R admin:admin /home/admin/.kube  (or)  mkdir /opt/autoteam/.kube  cp -r /etc/kubernetes/admin.conf /opt/autoteam/config  chown -R autoteam:autoteam /opt/autoteam/.kube |



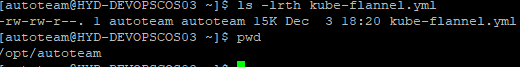
## Kubernetes networking model

##### **Deploy Flannel network**

<https://github.com/coreos/flannel/blob/master/Documentation/kube-flannel.yml>

For Kubernetes v1.7+

 kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml



Kubectl apply -f kube-flannel.yml

**Custer Join Command**

|  |
| --- |
| kubeadm token create --print-join-command |

**On Worker**

##### **Join the cluster**

Use the output from **kubeadm token create** command in previous step from the master server and run here

## Verifying the cluster

##### **Get Nodes status**

|  |
| --- |
| kubectl get nodes |

##### **Get component status**

|  |
| --- |
| kubectl get cs |
| kubectl get cs -o yaml |

kubectl taint nodes --all node-role.kubernetes.io/master-

kubectl get nodes -o json | jq .items[].spec.taints

Configure Registry username and password.

|  |
| --- |
| kubectl create secret docker-registry mydockercredentials --docker-server 192.168.136.87:5000 --docker-username admin --docker-password admin |

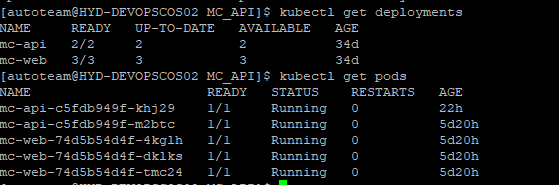
Application Controllers

* Deployment/statefulset/deamonset
* Configmap
* Services(clusterport,nodeport,load balancer)
* Ingress
* Horizontal Pod autotscaler

## Kubernetes Object: Deployment

**Deployments** represent a set of multiple, identical Pods with no unique identities. A **Deployment** runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive.. **Deployments** are managed by the **Kubernetes Deployment** controller.

|  |
| --- |
| Kubectl apply -f haepmautodeploy/eksctl/MC\_API/deployment.yaml  Kubectl get deployments |



## Kubernetes Object: Service

Service is an abstraction which defines a logical set of Pods and a policy by which to access. The set of Pods targeted by a Service is usually determined by a [selector](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/).

There are [four types](https://v1-13.docs.kubernetes.io/docs/concepts/services-networking/#publishing-services-service-types) of Kubernetes services:

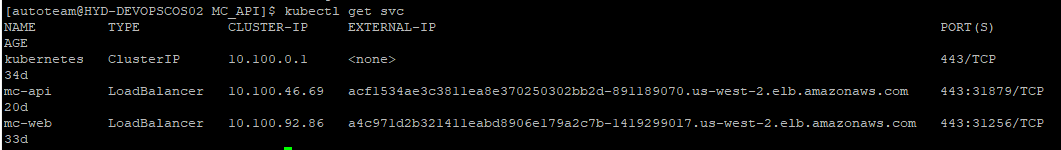
**ClusterIP.** This default type exposes the service on a cluster-internal IP. You can reach the service only from *within* the cluster.

**NodePort.** This type of service exposes the service on each node’s IP at a static port. A ClusterIP service is created automatically, and the NodePort service will route to it. From *outside* the cluster, you can contact the NodePort service by using “<NodeIP>:<NodePort>”.

**LoadBalancer.** This service type exposes the service externally using the load balancer of your cloud provider. The external load balancer routes to your NodePort and ClusterIP services, which are created automatically.

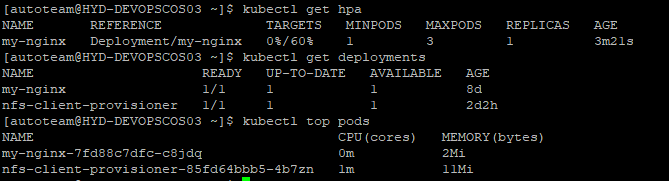
**ExternalName.** This type maps the service to the contents of the externalName field (e.g., foo.bar.example.com). It does this by returning a value for the CNAME record.

|  |
| --- |
| Kubectl apply -f haepmautodeploy/eksctl/MC\_API/service.yaml  Kubectl get svc |



**Horizontal pod autoscaler**

|  |
| --- |
| >kubectl autoscale deployment my-nginx --cpu-percent=60 --min=1 --max=3  kubectl describe hpa my-nginx |



If “Kubectl top pods” doesn’t give valid out put, one need to configure metrics-server(refer below for steps) or Heapster (deprecated)

**Ingress**

The concept of Ingress is split in two parts:

* The [Ingress Controller](https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/), it’s some kind of wrapper for an HTTP proxy,
* Ingress resources/rules that expose HTTP and HTTPS routes from outside the cluster to services within the cluster, depending on traffic rules.

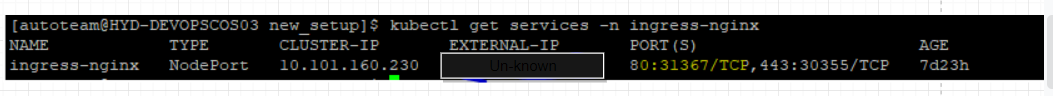
**Ingress Controller Installation**

Install a nginx ingress controller

> kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/static/mandatory.yaml

Ingress Controller is started, but not yet accessible externally from our K8s cluster. We need to create a NodePort Service to expose it to the outside world.

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/static/provider/baremetal/service-nodeport.yaml>



External-IP - <un-known>

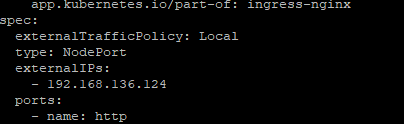
Access with any node Ip and port 31367 we will see a NGINX 404 page

Our **NGINX controller** is responding!

**Hack for local Dev Setup**

As we do not have any public Ip to be exposed as external-ip for the nginx ingress service controller

Update the external ip section in “**service-nodeport.yaml**”

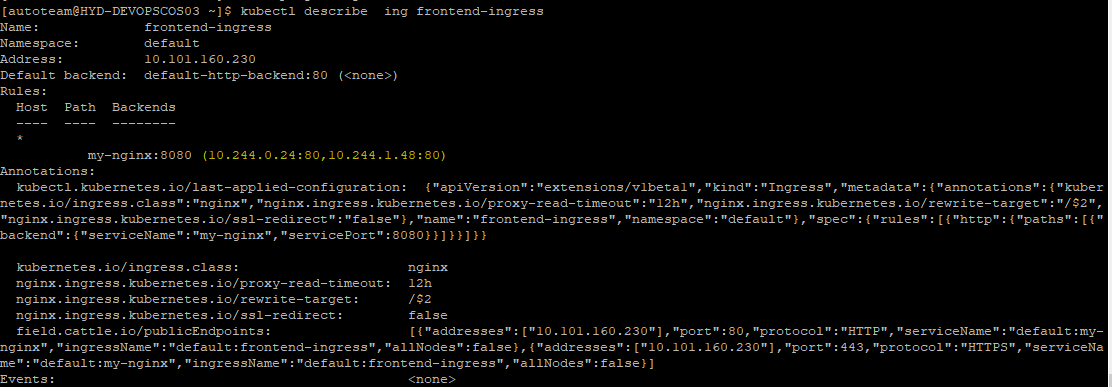




**Create Ingress to public expose application**

frontend-ingress.yaml

|  |
| --- |
| apiVersion: extensions/v1beta1  kind: Ingress  metadata:  annotations:  kubernetes.io/ingress.class: "nginx"  nginx.ingress.kubernetes.io/rewrite-target: /$2  nginx.ingress.kubernetes.io/proxy-read-timeout: "12h"  nginx.ingress.kubernetes.io/ssl-redirect: "false"  name: frontend-ingress  namespace: default  spec:  rules:  # - host: hyd-devopscos04.haicpm.com  - http:  paths:  - backend:  serviceName: my-nginx  servicePort: 8080 |





AS we added External Ip address to Ingress controller node port service now we can access ”my-nginx” application using external Ip

Reference :



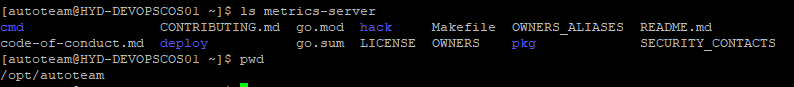
# **Kubernetes Metrics Server**

# Resource usage metrics, such as container CPU and memory usage, are available in Kubernetes through the Metrics API. These metrics can be either accessed directly by user, for example by using kubectl top command, or used by a controller in the cluster, e.g. Horizontal Pod Autoscaler, to make decisions.

* Through the **Metrics API** you can get the amount of resource currently used by a given node or a given pod
* **Metric server** is a cluster-wide aggregator of resource usage data

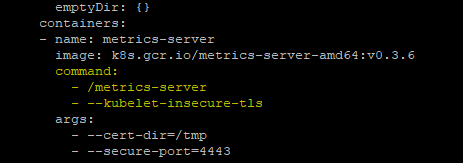


Refernce :

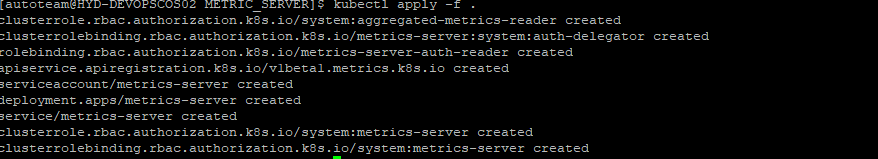


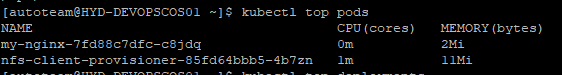
* git clone <https://github.com/kubernetes-sigs/metrics-server>
* >cd metrics-server/deploy/1.8+/

Update “metrics-server-deployment.yaml” as shown below to accept insecure tls



* kubectl create -f .





**For Kubernetes version** v1.17.3

**Issue:**

kubectl describe apiservice v1beta1.metrics.k8s.io

Status:

Conditions:

Last Transition Time: 2020-03-12T16:04:59Z

Message: failing or missing response from https://10.99.122.196:443/apis/metrics.k8s.io/v

1beta1: Get https://10.99.122.196:443/apis/metrics.k8s.io/v1beta1: net/http: request canceled while waiting

for connection (Client.Timeout exceeded while awaiting headers)

kubectl top pod

Error from server (ServiceUnavailable): the server is currently unable to handle the request (get pods.metrics.k8s.io)

**Reference:** <https://stackoverflow.com/questions/60531350/kubernetes-metrics-server-faileddiscoverycheck>

git clone https://github.com/kubernetes-sigs/metrics-server

$ cd metrics-server/deploy/kubernetes/

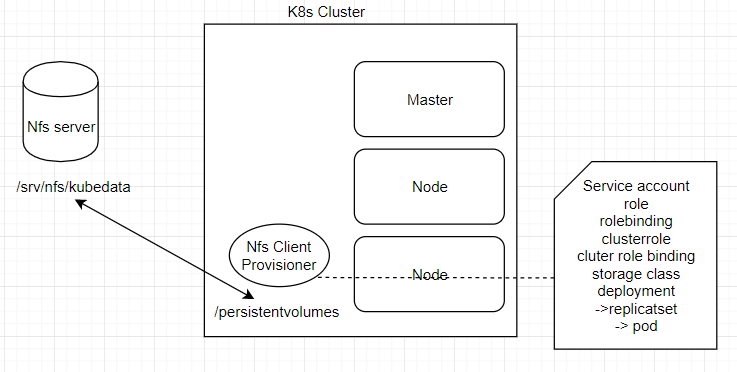
$ vi metrics-server-deployment.yaml

# **Dynamically provision NFS persistent Volumes in Kubernetes**

Kubernetes has **persistent volumes**. Persistent volumes are long-term storage in your Kubernetes cluster. Persistent volumes exist beyond containers, pods, and nodes .A pod uses a persistent volume claim to to get read and write access to the persistent volume.

**NFS Client Provisioner** creates a pod in k8’s cluster, which mounts /srv/nfs/kubedata to /persistentvolumes

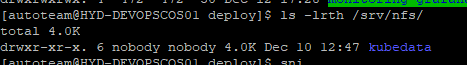
Also create a storage class and point to **NFS Client Provisioner pod**



**NFS-server setup**

>git clone <https://github.com/kubernetes-incubator/external-storage>

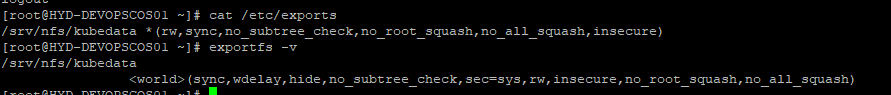
* >cd external-storage/nfs-client/deploy
* > mkdir –p /srv/nfs/kubedata ( can be on any external Machine)
* >sudo chown nobody: /srv/nfs/kubedata



* sudo yum install nfs-utils ( yum -y install nfs-common cifs-utils nfs4-acl-tools nfs-utils on all nodes)
* systemctl enable nfs-server
* systemctl start nfs-server (assume firewall disable)
* cat /etc/exports

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

* sudo exportfs -rav
* sudo exportfs -v



**On any worker node:**

* mount -t nfs 192.168.136.15:/srv/nfs/kubedata /mnt

(any issue :> yum -y install nfs-common cifs-utils nfs4-acl-tools nfs-utils on all nodes)

* >mount | grep kubedata
* >umount /mnt

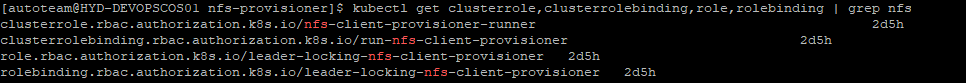
Verification successful

**Deploy client provisioner pod**

* git clone <https://github.com/justmeandopensource/kubernetes>
* cd Kubernetes/yamls/nfs-provisioner

->rbac.yaml – creates service account,role,rolebinding,clusterrole,clusterrole binding

* Kubectl create -f rbac.yml



->class.yaml – creates storage class

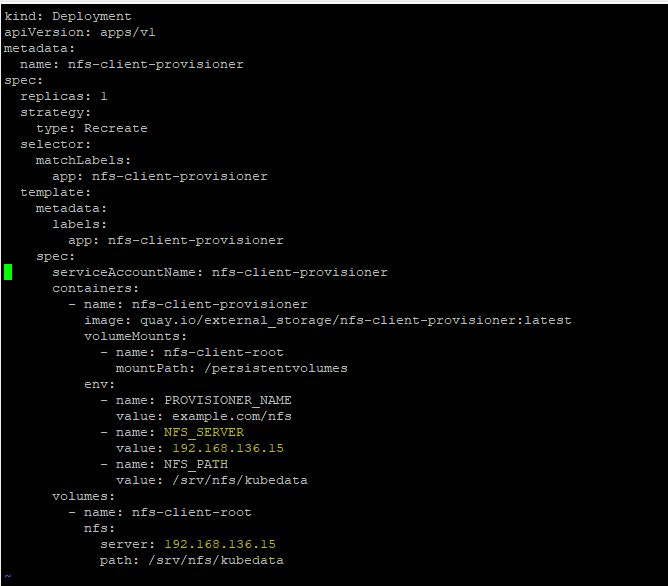
* kubectl create -f class.yml

( name will be there about storage class, it should be mentioned in persistent storage claim)

->deployment.yaml – create deploymet,replicatser

Update NfS server Ip as shown below

* kubectl create -f deployment.yml
* kubectl describe pod nfs-client-provisioner-85fd64bbb5-4b7zn

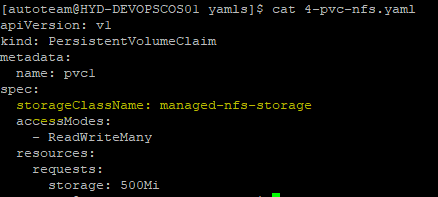


* kubectl get pv,pvc

no resource found

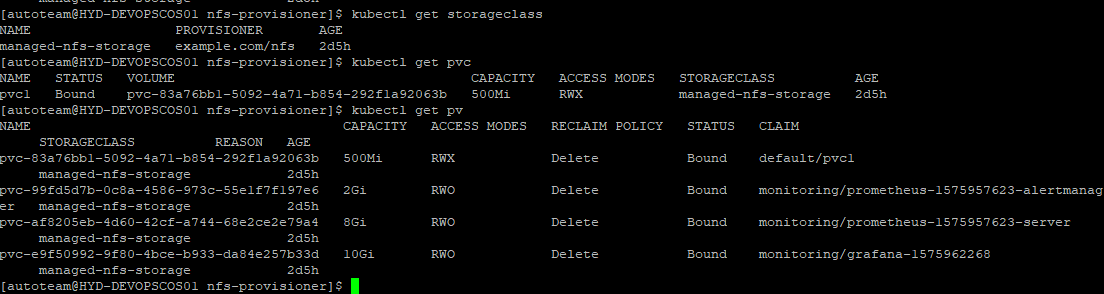
**Create persistent Volume claim(pvc)**

Storage class name show be updated as created earlier



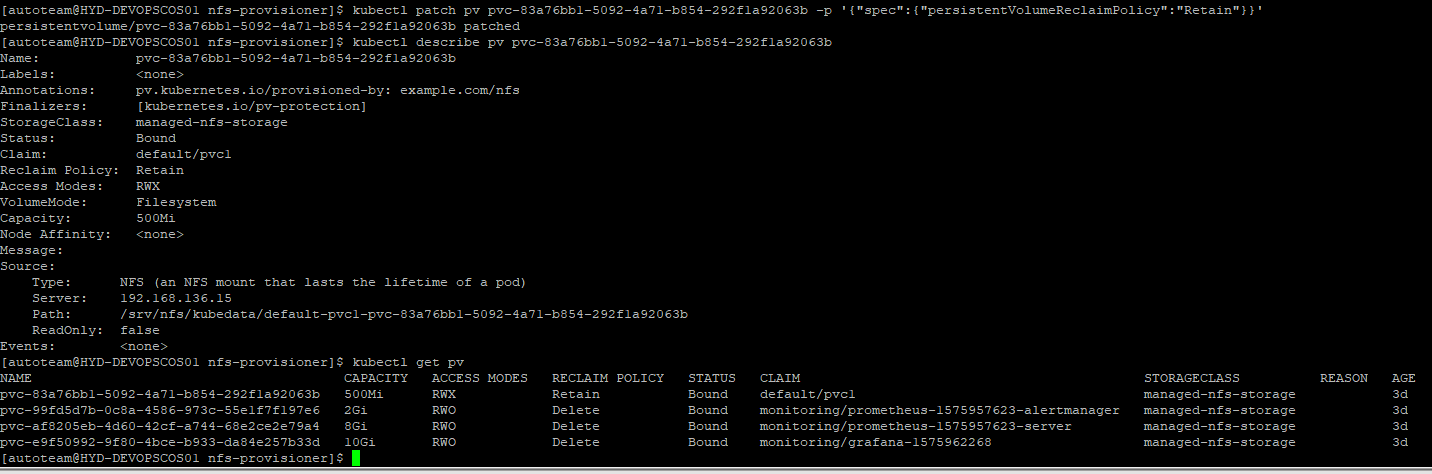
* Kubectl apply -f 4-pvc-nfs.yaml

Now pvc will trigger NFS provisioner and will create pv intern



**Change Recalim policy to Retain**

$ kubectl patch pv <pv name> -p '{"spec":{"persistentVolumeReclaimPolicy":"Retain"}}'



**Create pod and consume persistent Volume**

deployment.yaml

|  |
| --- |
| apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2  kind: Deployment  metadata:  name: my-nginx  spec:  selector:  matchLabels:  app: my-nginx  replicas: 2 # tells deployment to run 2 pods matching the template  template:  metadata:  labels:  app: my-nginx  spec:  volumes:  - name: host-volume  persistentVolumeClaim:  claimName: pvc1  containers:  - name: my-nginx  image: nginx:1.7.9  resources:  requests:  # memory: "${vis\_memory}"  cpu: "10m"  limits:  # memory: "${vis\_memory\_limit}"  cpu: "100m"  ports:  - containerPort: 80  volumeMounts:  - mountPath: /var/log/nginx/  name: host-volume |

# <https://kubernetes.io/docs/concepts/services-networking/add-entries-to-pod-etc-hosts-with-host-aliases/>

apiVersion: v1

kind: Pod

metadata:

name: hostaliases-pod

spec:

restartPolicy: Never

hostAliases:

- ip: "127.0.0.1"

hostnames:

- "foo.local"

- "bar.local"

- ip: "10.1.2.3"

hostnames:

- "foo.remote"

- "bar.remote"

containers:

- name: cat-hosts

image: busybox

command:

- cat

args:

- "/etc/hosts"

Assuming Kubernetes cluster is already setup and Metric server is installed.

|  |  |
| --- | --- |
| **Type** | **IP** |
| Kubernetes master/node1 | 192.168.136.16 |
| Node 2 | 192.168.136.20 |
| Node3 | 192.168.136.47 |
| NFS | 192.168.136.83 |
| Registry | 192.168.136.87 |

# **Monitoring using Prometheus and Grafana**

# **Kubernetes Creating Users and Roles**

Create user for DEV, kubernetes not aware of any user and groups it only cares about certificates. If we provide valid certificate during authentication will be allowed to access the cluster. Depending up on the roles u define in cluster a user can do specific things like cluster-access or access to specific name space

User : DEV namespace: Development

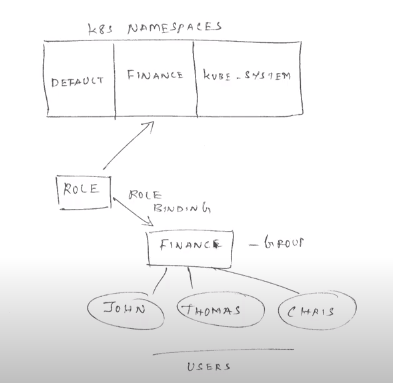
We need to create certificated – dev.key(private key), dev.csr(certificate sign request),dev.crt(certificate).

We require kubernetes to sign user (dev) certificate

ca.key(privatekey) ,ca.crt (/etc/kubernetes/pki)

In order for DEV user to contact Kubectl it requires dev.key(private key), dev.crt

A role will be created, and bind role to specific user. Role also give access to specific namespace.



Steps

* kubectl create ns development

#create certificate for user dev

• openssl genrsa -out dev.key 2048

#create certificate signing request - member of group development

• openssl req -new -key dev.key -out dev.csr -subj "/CN=dev/O=devlopment"

#sign key with kubernetes service account

• openssl x509 -req -in dev.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out dev.crt -days 3650

**Create a kubeconfig file with ca.cer, dev.key and dev.cer**

#create certificate for user dev

• openssl genrsa -out dev.key 2048

#create certificate signing request - member of group development

• openssl req -new -key dev.key -out dev.csr -subj "/CN=dev/O=devlopment"

#sign key with kubernetes service account

• openssl x509 -req -in dev.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out dev.crt -days 3650

https://192.168.136.16:6443

kubectl --kubeconfig=dev.kubeconfig config set-cluster kubernetes --server https://192.168.136.16:6443 --certificate-authority=ca.crt

Add user to kubeconfig

kubectl --kubeconfig=dev.kubeconfig config set-credentials dev --client-certificate /home/admin/users/dev.crt --client-key /home/admin/users/dev.key

set the context

kubectl --kubeconfig=dev.kubeconfig config set-context dev-kubernetes --cluster kubernetes --namespace development --user dev

update dev.kubeconfig current-context to dev-kubernetes

[dev@HYD-DEVOPSMC01 home]$ cd /home/dev/.kube/

[dev@HYD-DEVOPSMC01 .kube]$ ls

cache ca.crt config dev.crt dev.key http-cache

[dev@HYD-DEVOPSMC01 .kube]$ ls -lrth

total 20K

-rw------- 1 dev dev 382 May 28 09:31 config

-rw-r--r-- 1 dev dev 1.7K May 28 09:34 dev.key

-rw-r--r-- 1 dev dev 997 May 28 09:34 dev.crt

-rw-r--r-- 1 dev dev 1.1K May 28 09:34 ca.crt

drwxr-x--- 3 dev dev 23 May 28 09:35 cache

**Jenkins linux slave setup in windows master**

ssh-keygen

[root@HYD-DEVOPSCOS01 .ssh]# java -version

java version "1.8.0\_91"

[root@HYD-DEVOPSCOS01 .ssh]# echo $JAVA\_HOME

/usr/java/jdk1.8.0\_91

[root@HYD-DEVOPSCOS01 .ssh]# echo $JRE\_HOME

/usr/java/jdk1.8.0\_91/jre