Hướng dẫn cài Kubernetes cluster

[**Tài nguyên, kiến trúc hệ thống**](#_9ls69ox474t0) **2**

[Thông tin địa chỉ ip server cài đặt k8s](#_lk9a7e25vw1z) 2

[Mô hình Kubernetes cluster](#_8af8eh7azwxr) 3

[Kiến trúc hệ thống giám sát prometheus](#_xpmmuwmderdo) 3

[Log tập trung](#_1ivvn0r32g3v) 4

[**Cài đặt docker**](#_iy2hb18kf1pu) **5**

[Kiểm tra phiên bản của centos](#_gambkxfcu2gl) 5

[Setup daemon](#_c5nhrufxorqv) 6

[**Cài đặt K8s**](#_r7ompkuk3rz) **7**

[Cài đặt Kubeadm, Kubelet, Kubectl, trên các server triển khai k8s](#_wy01ntq83zkn) 7

[Download các package/images.](#_7d7ml6h7p3xz) 7

[Cài đặt kubeadm, kubelet, kubectl](#_ckkxt3g24inb) 8

[Cài đặt etcd cluster](#_kll2fzb02uex) 10

[Cài đặt haproxy](#_eg5aeuso2lp5) 16

[Cài đặt HA cluster với kubeadm](#_35gnhobgqe0c) 17

[**Cài đặt rancher**](#_frfjzrj9zy34) **23**

[Download image để cài đặt rancher](#_v30xbpptmyad) 23

[Cài đặt racher](#_vhgast29f2zq) 23

[**Cài đặt prometheus**](#_1pzwlsr9elkk) **25**

[Download những images cần thiết](#_v4ej8xca2g4j) 25

[Cài đặt localpath làm storage cho prometheus server](#_x8k1xb33a824) 25

[Tạo các file yaml để deploy prometheus](#_h3i9gp9w8zeh) 26

[Triển khai prometheus](#_lmh725zb0fmt) 32

[**Cài đặt grafana**](#_ymir9ezb1dvx) **32**

[Download những images cần thiết](#_lnrrzulpls6m) 32

[Triển khai grafana](#_vw6bcyisvm81) 33

[**Triển khai log tập trung**](#_vqftfjwj04e) **33**

[Download images cần thiết.](#_yl6xdaz3q1mw) 33

[Cấu hình file yaml](#_bs5u14wjdgdk) 33

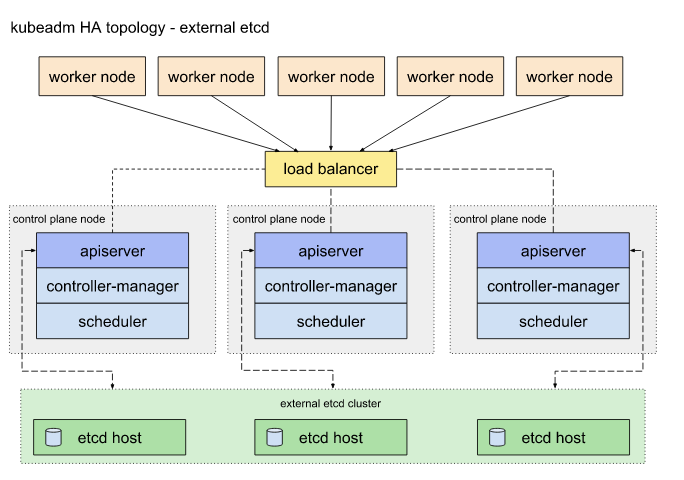
# Tài nguyên, kiến trúc hệ thống

## Thông tin địa chỉ ip server cài đặt k8s

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STT** | **Cloud** | **Tên dự án** | **Service** | **Hostname** | **IP** | |
| **DCN** | **IP Khác** |
| 1 | Cisco\_2020\_HLC | VDA | ETCD - K8S | etcd\_k8s\_01 | 10.207.58.1/24 |  |
| 2 | Cisco\_2020\_HLC | VDA | ETCD - K8S | etcd\_k8s\_02 | 10.207.58.2/24 |  |
| 3 | Cisco\_2020\_HLC | VDA | ETCD - K8S | etcd\_k8s\_03 | 10.207.58.3/24 |  |
| 4 | Cisco\_2020\_HLC | VDA | Master - K8S | master\_k8s\_01 | 10.207.58.4/24 |  |
| 5 | Cisco\_2020\_HLC | VDA | Master - K8S | master\_k8s\_02 | 10.207.58.5/24 |  |
| 6 | Cisco\_2020\_HLC | VDA | Master - K8S | master\_k8s\_03 | 10.207.58.6/24 |  |
| 7 | Cisco\_2020\_HLC | VDA | worker - K8S | worker\_k8s\_01 | 10.207.58.7/24 |  |
| 8 | Cisco\_2020\_HLC | VDA | worker - K8S | worker\_k8s\_02 | 10.207.58.8/24 |  |
| 9 | Cisco\_2020\_HLC | VDA | worker - K8S | worker\_k8s\_03 | 10.207.58.9/24 |  |
| 10 | Cisco\_2020\_HLC | VDA | worker - K8S | worker\_k8s\_04 | 10.207.58.10/24 |  |
| 11 | Cisco\_2020\_HLC | VDA | worker - K8S | worker\_k8s\_05 | 10.207.58.11/24 |  |
| 12 | Cisco\_2020\_HLC | VDA | worker - K8S | worker\_k8s\_06 | 10.207.58.12/24 |  |

## Mô hình Kubernetes cluster

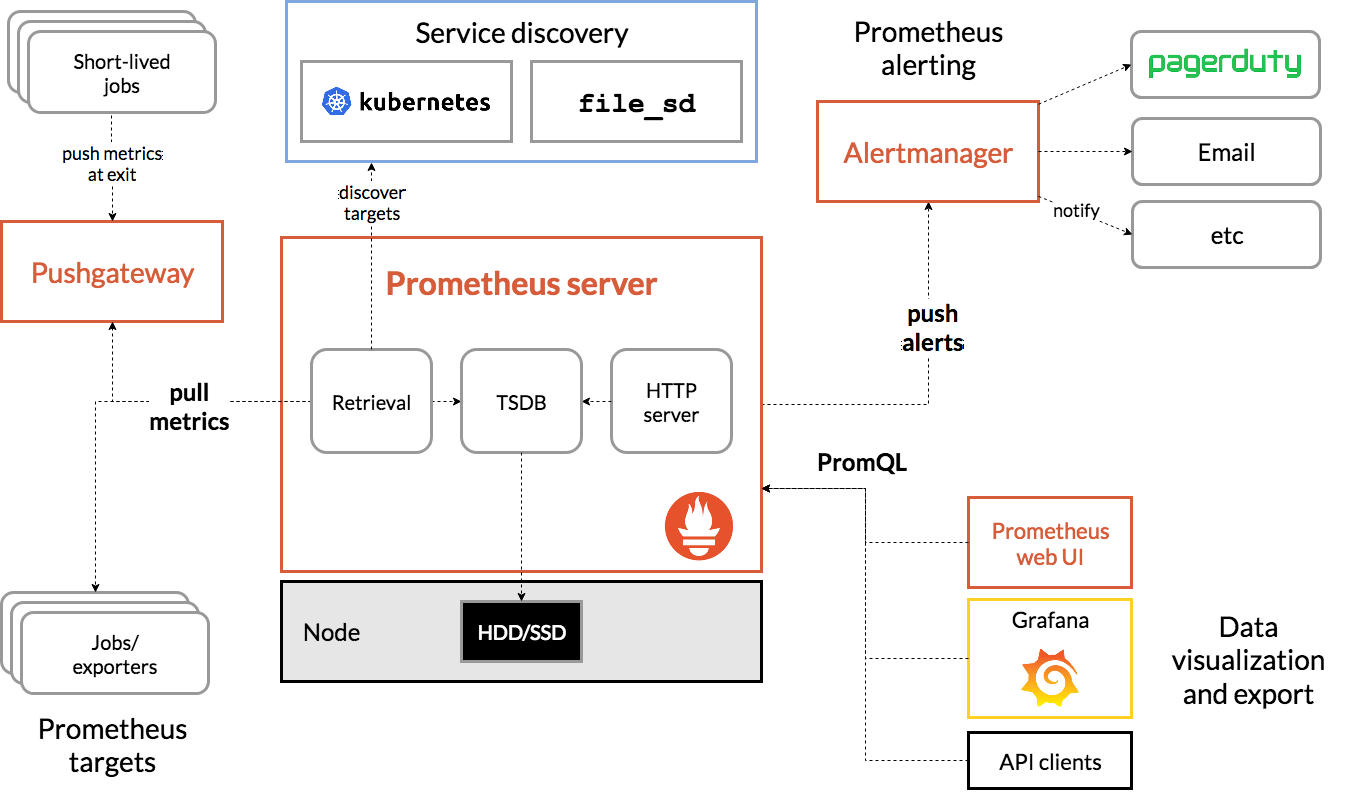
K8s được triển khai theo mô hình sau



Trong đó có 3 node master, 3 node cài etcd, 6 node làm worker, sử dụng 1 node etcd để cài haproxy để làm loadbalance như mô hình trên.

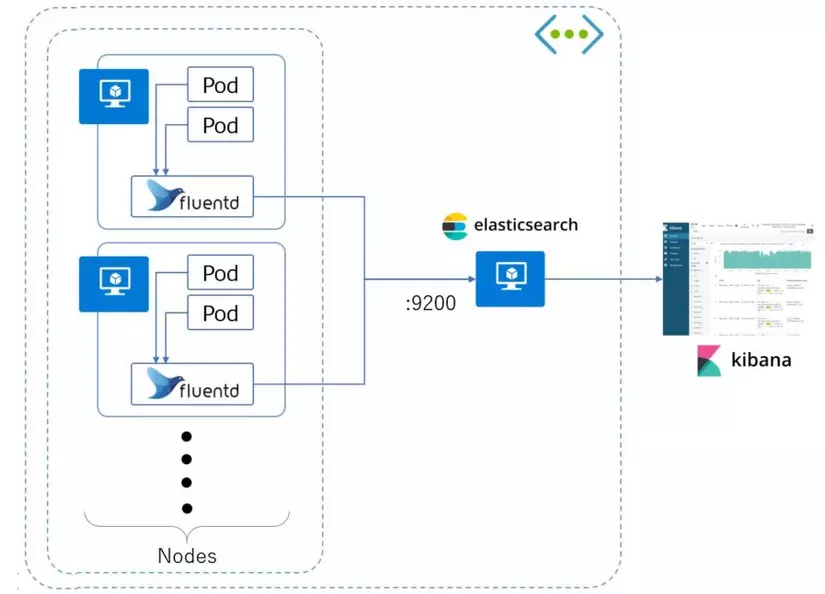
## Kiến trúc hệ thống giám sát prometheus

Sử dụng prometheus để thu thập log metrics của hệ thống, và cảnh báo khi xảy ra bất thường



## Log tập trung

Sử dụng fluentd để thu thập log ứng dụng đẩy vào elasticsearch và sử dụng kibana để visualize dữ liệu



# Cài đặt docker

Phiên bản cài đặt Docker version 19.03.15

## Kiểm tra phiên bản của centos

|  |
| --- |
| [centos@ip-172-31-42-168 ~]$ cat /etc/centos-release CentOS Linux release 7.7.1908 (Core) |

Vào đường link <https://download.docker.com/linux/centos/> chọn phiên bản của centos, sau đó vào đường dẫn x86\_64/stable/Packages/ và tải xuống phiên bản docker cần cài đặt.

* [containerd.io-1.4.4-3](https://download.docker.com/linux/centos/7/x86_64/stable/Packages/containerd.io-1.4.4-3.1.el7.x86_64.rpm)
* [docker-ce-cli-19.03.15](https://download.docker.com/linux/centos/7/x86_64/stable/Packages/docker-ce-cli-19.03.15-3.el7.x86_64.rpm)
* [docker-ce-19.03.15](https://download.docker.com/linux/centos/7/x86_64/stable/Packages/docker-ce-19.03.15-3.el7.x86_64.rpm)
* [libtool-ltdl](http://mirror.centos.org/centos/7/os/x86_64/Packages/libtool-ltdl-2.4.2-22.el7_3.x86_64.rpm)
* [pigz-2.3.3-1](http://mirror.centos.org/centos/7/extras/x86_64/Packages/pigz-2.3.3-1.el7.centos.x86_64.rpm)
* [container-selinux-2.119](http://mirror.centos.org/centos/7/extras/x86_64/Packages/container-selinux-2.119.2-1.911c772.el7_8.noarch.rpm)

Sau đó scp tất cả file .rpm lên từng server và thực hiện cài đặt như sau

Sau đó thực hiện cài đặt

|  |
| --- |
| yum install libtool-ltdl-2.4.2-22.el7\_3.x86\_64.rpm  yum install pigz-2.3.3-1.el7.centos.x86\_64.rpm  yum install container-selinux-2.119.2-1.911c772.el7\_8.noarch.rpm  yum install containerd.io-1.4.3-3.2.el7.x86\_64.rpm  yum install docker-ce-cli-19.03.15-3.el7.x86\_64.rpm  yum install docker-ce-19.03.15-3.el7.x86\_64.rpm |

## Setup daemon

|  |
| --- |
| cat > /etc/docker/daemon.json <<EOF {  "exec-opts": ["native.cgroupdriver=systemd"],  "log-driver": "json-file",  "log-opts": {  "max-size": "100m"  },  "storage-driver": "overlay2",  "storage-opts": [  "overlay2.override\_kernel\_check=true"  ] } EOF |

Start docker bằng câu lệnh sau

|  |
| --- |
| sudo systemctl start docker |

Cấp quyền cho user sử dụng docker

|  |
| --- |
| sudo groupadd docker sudo usermod -aG docker ${USER} |

Sau khi cài đặt xong kiểm tra phiên bản của docker như sau

|  |
| --- |
| [root@ads-m1 centos]# docker --version Docker version 19.03.15, build 99e3ed8919 |

# Cài đặt K8s

## Cài đặt Kubeadm, Kubelet, Kubectl, trên các server triển khai k8s

### Download các package/images.

Sử dụng yumdownloader để download các packages ở máy local sau đó scp lên server.

**Thực hiện các bước ở máy local (có kết nối mạng) như sau**

* Add **Kubernetes (K8s)** 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 |

* Build yum cache.

|  |
| --- |
| **#** yum makecache fast |

* Tạo folder để download các packages của K8s

|  |
| --- |
| **#** mkdir ~/k8s **#** cd ~/k8s |

* Download Kubernetes (K8s) packages sử dụng **yumdownloader**.

|  |
| --- |
| **#** yumdownloader --resolve kubelet kubeadm kubectl |

* Download Images để khởi tạo k8s

|  |
| --- |
| **#** docker pull k8s.gcr.io/kube-apiserver:v1.14.1 **#** docker pull k8s.gcr.io/kube-controller-manager:v1.14.1 **#** docker pull k8s.gcr.io/kube-scheduler:v1.14.1 **#** docker pull k8s.gcr.io/kube-proxy:v1.14.1 **#** docker pull k8s.gcr.io/pause:3.1 **#** docker pull k8s.gcr.io/etcd:3.3.10 **#** docker pull k8s.gcr.io/coredns:1.3.1  **#** docker pull k8s.gcr.io/etcd:3.4.3-0 |

* Export các images liên quan đến k8s vào folder k8s

|  |
| --- |
| **#** docker save k8s.gcr.io/kube-apiserver:v1.14.1 > ~/k8s/kube-apiserver.tar **#** docker save k8s.gcr.io/kube-controller-manager:v1.14.1 > ~/k8s/kube-controller-manager.tar **#** docker save k8s.gcr.io/kube-scheduler:v1.14.1 > ~/k8s/kube-scheduler.tar **#** docker save k8s.gcr.io/kube-proxy:v1.14.1 > ~/k8s/kube-proxy.tar **#** docker save k8s.gcr.io/pause:3.1 > ~/k8s/pause.tar **#** docker save k8s.gcr.io/etcd:3.3.10 > ~/k8s/etcd.tar **#** docker save k8s.gcr.io/coredns:1.3.1 > ~/k8s/coredns.tar  **#** docker save k8s.gcr.io/etcd:3.4.3-0 > ~/k8s/etcd.tar |

* Sử dụng **Calico** làm network cho k8s.

|  |
| --- |
| **#** cd ~/k8s **#** wget https://docs.projectcalico.org/v3.10/manifests/calico.yaml |

Chuyển toàn bộ folder k8s ở local lên các server cài đặt sử dụng scp

### Cài đặt kubeadm, kubelet, kubectl

* 1. Cấu hình các tham số
* Set các tham số forward network như sau

|  |
| --- |
| **#** cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf br\_netfilter EOF  cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf net.bridge.bridge-nf-call-ip6tables = 1 net.bridge.bridge-nf-call-iptables = 1 EOF **#** sudo sysctl --system |

* Bật mode bridged traffic trên iptables

Check mode br\_netfilter như sau: lsmod | grep br\_netfilter

Nếu chưa bật thì gọi lệnh sau để bật: sudo modprobe br\_netfilter

* Tắt SELinux

|  |
| --- |
| sudo setenforce 0 sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config |

* Tắt swap

|  |
| --- |
| **#** swapoff -a **#** sed -e '/swap/s/^/#/g' -i /etc/fstab |

* 1. Cài đặt trên từng server

Vào thư mục chứa các package/images được copy từ local lên thực hiện các bước sau

* Cài đặt k8s từ các package

|  |
| --- |
| **#** rpm -ivh --replacefiles --replacepkgs ~/k8s/\*.rpm |

* Enable bash completion for **kubectl**.

**#** source <(kubectl completion bash)  
**#** kubectl completion bash > /etc/bash\_completion.d/kubectl

* Import các file .tar của các images vào Docker

|  |
| --- |
| **#** cd ~/k8s  **#** docker load < coredns.tar **#** docker load < kube-proxy.tar **#** docker load < etcd.tar **#** docker load < kube-scheduler.tar **#** docker load < kube-apiserver.tar **#** docker load < pause.tar **#** docker load < kube-controller-manager.tar  **#** docker load < etcd.tar |

Sau khi chạy xong kiểm tra phiên bản của kubelet

|  |
| --- |
| **#** kubelet --version Kubernetes v1.21.2 |

## Cài đặt etcd cluster

* Step1: Cấu hình kubelet để quản lý etcd, thực hiện trên toàn bộ hosts cài đặt etcd

|  |
| --- |
| cat << EOF > /etc/systemd/system/kubelet.service.d/20-etcd-service-manager.conf [Service] ExecStart= **#** Replace "systemd" with the cgroup driver of your container runtime. The default value in the kubelet is "cgroupfs". ExecStart=/usr/bin/kubelet --address=127.0.0.1 --pod-manifest-path=/etc/kubernetes/manifests --cgroup-driver=systemd Restart=always EOF  systemctl daemon-reload systemctl restart kubelet |

Sau khi chạy xong thực hiện kiểm tra trạng thái của kubelet như sau:

|  |
| --- |
| **#** systemctl status kubelet  kubelet.service - kubelet: The Kubernetes Node Agent  Loaded: loaded (/usr/lib/systemd/system/kubelet.service; disabled; vendor preset: disabled)  Drop-In: /usr/lib/systemd/system/kubelet.service.d  └─10-kubeadm.conf  /etc/systemd/system/kubelet.service.d  └─20-etcd-service-manager.conf  Active: active (running) since CN 2021-07-11 16:05:39 UTC; 10s ago  Docs: https://kubernetes.io/docs/  Main PID: 19487 (kubelet)  Tasks: 14  Memory: 28.0M  CGroup: /system.slice/kubelet.service  └─19487 /usr/bin/kubelet --address=127.0.0.1 --pod-manifest-path=/etc/kubernetes/manifests --cgroup-driver=systemd |

* Lưu ý lúc kiểm tra có thể có lỗi như sau

|  |
| --- |
| Failed to run kubelet" err="failed to run Kubelet: misconfiguration: kubelet cgroup driver: \"systemd\" is different from docker cgroup driver: \"cgroupfs\"" |

=> Thực hiện cấu hình docker như phần trước rồi chạy

sudo systemctl restart docker

* Step2: Chạy tập lệnh dưới trên mỗi host để thực hiện cấu hình cấu hình etcd cho kubeadm trên từng host, thực hiện trên 3 node (10.207.58.1, 10.207.58.2, 10.207.58.3)

|  |
| --- |
| export HOST0=10.207.58.1 export HOST1=10.207.58.2 export HOST2=10.207.58.3  # Create temp directories to store files that will end up on other hosts. mkdir -p /tmp/${HOST0}/ /tmp/${HOST1}/ /tmp/${HOST2}/  ETCDHOSTS=(${HOST0} ${HOST1} ${HOST2}) NAMES=("infra0" "infra1" "infra2")  for i in "${!ETCDHOSTS[@]}"; do HOST=${ETCDHOSTS[$i]} NAME=${NAMES[$i]} cat << EOF > /tmp/${HOST}/kubeadmcfg.yaml apiVersion: "kubeadm.k8s.io/v1beta2" kind: ClusterConfiguration etcd:  local:  serverCertSANs:  - "${HOST}"  peerCertSANs:  - "${HOST}"  extraArgs:  initial-cluster: ${NAMES[0]}=https://${ETCDHOSTS[0]}:2380,${NAMES[1]}=https://${ETCDHOSTS[1]}:2380,${NAMES[2]}=https://${ETCDHOSTS[2]}:2380  initial-cluster-state: new  name: ${NAME}  listen-peer-urls: https://${HOST}:2380  listen-client-urls: https://${HOST}:2379  advertise-client-urls: https://${HOST}:2379  initial-advertise-peer-urls: https://${HOST}:2380 EOF done |

* Step3: Tạo certificate

Thực hiện tọa certs trên 1 host, thực hiện trên $HOST0 có ip như trên, chạy câu lệnh sau:

|  |
| --- |
| **#** kubeadm init phase certs etcd-ca [certs] Generating "etcd/ca" certificate and key |

Sau khi chạy xong sẽ tạo ra 2 file ở đường dẫn như sau

|  |
| --- |
| /etc/kubernetes/pki/etcd/ca.crt /etc/kubernetes/pki/etcd/ca.key |

Sau khi tạo certs xong thì copy 2 file trên đến các host cài etcd, đúng đường dẫn như trên

* Step4: Tạo certificates cho từng host

|  |
| --- |
| kubeadm init phase certs etcd-server --config=/tmp/${HOST2}/kubeadmcfg.yaml kubeadm init phase certs etcd-peer --config=/tmp/${HOST2}/kubeadmcfg.yaml kubeadm init phase certs etcd-healthcheck-client --config=/tmp/${HOST2}/kubeadmcfg.yaml kubeadm init phase certs apiserver-etcd-client --config=/tmp/${HOST2}/kubeadmcfg.yaml cp -R /etc/kubernetes/pki /tmp/${HOST2}/ **#** cleanup non-reusable certificates find /etc/kubernetes/pki -not -name ca.crt -not -name ca.key -type f -delete  kubeadm init phase certs etcd-server --config=/tmp/${HOST1}/kubeadmcfg.yaml kubeadm init phase certs etcd-peer --config=/tmp/${HOST1}/kubeadmcfg.yaml kubeadm init phase certs etcd-healthcheck-client --config=/tmp/${HOST1}/kubeadmcfg.yaml kubeadm init phase certs apiserver-etcd-client --config=/tmp/${HOST1}/kubeadmcfg.yaml cp -R /etc/kubernetes/pki /tmp/${HOST1}/ find /etc/kubernetes/pki -not -name ca.crt -not -name ca.key -type f -delete  kubeadm init phase certs etcd-server --config=/tmp/${HOST0}/kubeadmcfg.yaml kubeadm init phase certs etcd-peer --config=/tmp/${HOST0}/kubeadmcfg.yaml kubeadm init phase certs etcd-healthcheck-client --config=/tmp/${HOST0}/kubeadmcfg.yaml kubeadm init phase certs apiserver-etcd-client --config=/tmp/${HOST0}/kubeadmcfg.yaml **#** No need to move the certs because they are for HOST0  **#** clean up certs that should not be copied off this host find /tmp/${HOST2} -name ca.key -type f -delete find /tmp/${HOST1} -name ca.key -type f -delete |

* Step5: Copy certs và cấu hình kubeadm đến từng host tương ứng

|  |
| --- |
| USER=centos HOST=${HOST1} scp -r /tmp/${HOST}/\* ${USER}@${HOST}: ssh ${USER}@${HOST} USER@HOST $ sudo -Es root@HOST $ chown -R root:root pki root@HOST $ mv pki /etc/kubernetes/  thực hiện tương tự cho $HOST2 |

* Step6: Kiểm tra lại thông tin các file phải đầy đủ như sau

$HOST0

|  |
| --- |
| /tmp/${HOST0} └── kubeadmcfg.yaml --- /etc/kubernetes/pki ├── apiserver-etcd-client.crt ├── apiserver-etcd-client.key └── etcd  ├── ca.crt  ├── ca.key  ├── healthcheck-client.crt  ├── healthcheck-client.key  ├── peer.crt  ├── peer.key  ├── server.crt  └── server.key |

|  |
| --- |
| **$**HOME └── kubeadmcfg.yaml --- /etc/kubernetes/pki ├── apiserver-etcd-client.crt ├── apiserver-etcd-client.key └── etcd  ├── ca.crt  ├── healthcheck-client.crt  ├── healthcheck-client.key  ├── peer.crt  ├── peer.key  ├── server.crt  └── server.key |

$HOST2

|  |
| --- |
| **$**HOME └── kubeadmcfg.yaml --- /etc/kubernetes/pki ├── apiserver-etcd-client.crt ├── apiserver-etcd-client.key └── etcd  ├── ca.crt  ├── healthcheck-client.crt  ├── healthcheck-client.key  ├── peer.crt  ├── peer.key  ├── server.crt  └── server.key |

* Step7: Trên mỗi host cài đặt etcd tạo các pod manifests như sau

|  |
| --- |
| **#** Thực hiện trên HOST0  root@HOST0 $ kubeadm init phase etcd local --config=/tmp/${HOST0}/kubeadmcfg.yaml  **#** Thực hiện trên HOST1 root@HOST1 $ kubeadm init phase etcd local --config=/tmp/${HOST1}/kubeadmcfg.yaml **#** Thực hiện trên HOST2  root@HOST2 $ kubeadm init phase etcd local --config=/tmp/${HOST2}/kubeadmcfg.yaml |

* Step8: Kiểm tra trạng thái của etcd cluster

|  |
| --- |
| docker run --rm -it \ --net host \ -v /etc/kubernetes:/etc/kubernetes k8s.gcr.io/etcd:3.4.3-0 etcdctl \ --cert /etc/kubernetes/pki/etcd/peer.crt \ --key /etc/kubernetes/pki/etcd/peer.key \ --cacert /etc/kubernetes/pki/etcd/ca.crt \ --endpoints https://${HOST0}:2379 endpoint health --cluster |

## Cài đặt haproxy

Phần này sẽ sử dụng 1 node của etcd để cài haproxy, chọn node $HOST0

Thực hiện download haproxy từ local như sau

|  |
| --- |
| wget http://mirror.centos.org/centos/7/updates/x86\_64/Packages/haproxy-1.5.18-9.el7\_9.1.x86\_64.rpm |

Sau đó scp file haproxy-1.5.18-9.el7\_9.1.x86\_64.rpm lên server cài haproxy và thực hiện câu lệnh như sau

|  |
| --- |
| sudo yum install haproxy-1.5.18-9.el7\_9.1.x86\_64.rpm |

Sau đó thực hiện khởi động haproxy và kiểm tra trạng thái của nó như sau

|  |
| --- |
| **#** systemctl restart  **#** systemctl status haproxy  ● haproxy.service - HAProxy Load Balancer  Loaded: loaded (/usr/lib/systemd/system/haproxy.service; disabled; vendor preset: disabled)  Active: active (running) since CN 2021-07-11 17:46:43 UTC; 2min 5s ago  Main PID: 9690 (haproxy-systemd)  Tasks: 3  Memory: 1.9M  CGroup: /system.slice/haproxy.service  ├─9690 /usr/sbin/haproxy-systemd-wrapper -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid  ├─9691 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -Ds  └─9692 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -Ds  Th07 11 17:46:43 ads-e1 systemd[1]: Started HAProxy Load Balancer.  Th07 11 17:46:43 ads-e1 haproxy-systemd-wrapper[9690]: haproxy-systemd-wrapper: executing /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /...pid -Ds  Hint: Some lines were ellipsized, use -l to show in full. |

## Cài đặt HA cluster với kubeadm

* Copy các file sau từ 1 trong các node của etcd vào 1 node của master (sử dụng node master\_k8s\_01 có ip 10.207.58.4)

|  |
| --- |
| export CONTROL\_PLANE="centos@master\_k8s\_01" scp /etc/kubernetes/pki/etcd/ca.crt "${CONTROL\_PLANE}": scp /etc/kubernetes/pki/apiserver-etcd-client.crt "${CONTROL\_PLANE}": scp /etc/kubernetes/pki/apiserver-etcd-client.key "${CONTROL\_PLANE}": |

Sau đó ssh vào server master\_k8s\_01 chuyển các file vừa copy từ node etcd đến thư mục tương ứng như sau

|  |
| --- |
| /etc/kubernetes/pki/etcd/ca.crt  /etc/kubernetes/pki/apiserver-etcd-client.crt /etc/kubernetes/pki/apiserver-etcd-client.key |

* Cấu hình Load balancer cho phần cấu hình controlPlaneEndpoint

Thực hiện ssh vào server cài haproxy và thêm vào file cấu hình /etc/haproxy/haproxy.cfg như sau

|  |
| --- |
| frontend kubernetes-frontend  bind 10.207.58.1:6443  mode tcp  option tcplog  default\_backend kubernetes-backend  backend kubernetes-backend  mode tcp  option tcp-check  balance roundrobin  server kmaster1 10.207.58.4:6443 check fall 3 rise 2  server kmaster2 10.207.58.5:6443 check fall 3 rise 2  server kmaster3 10.207.58.6:6443 check fall 3 rise 2 |

Sau đó restart lại haproxy như sau

|  |
| --- |
| systemctl restart haproxy |

* Tiếp theo vào server master\_k8s\_01 tạo 1 file cấu hình kubeadm như sau:

|  |
| --- |
| apiVersion: kubeadm.k8s.io/v1beta2 kind: ClusterConfiguration kubernetesVersion: stable controlPlaneEndpoint: "10.207.58.1:6443" etcd:  external:  endpoints:  - https://10.207.58.1:2379  - https://10.207.58.2:2379  - https://10.207.58.3:2379  caFile: /etc/kubernetes/pki/etcd/ca.crt  certFile: /etc/kubernetes/pki/apiserver-etcd-client.crt  keyFile: /etc/kubernetes/pki/apiserver-etcd-client.key |

* Khởi tạo kubeadm như sau

|  |
| --- |
| **#** kubeadm init --config kubeadm-config.yaml --upload-certs Your Kubernetes control-plane has initialized successfully!  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  Alternatively, if you are the root user, you can run:   export KUBECONFIG=/etc/kubernetes/admin.conf  You should now deploy a pod network to the cluster. Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:  https://kubernetes.io/docs/concepts/cluster-administration/addons/  You can now join any number of the control-plane node running the following command on each as root:   kubeadm join 172.31.10.115:6443 --token vlkjp4.l71xfzz6woofctbv \  --discovery-token-ca-cert-hash sha256:4ec7dd568e841323c0f6c636045fc3e264f1562c95e6e5b5cd83fbffce2c0166 \  --control-plane --certificate-key 9c541d57ed688f9a65394455161e899eaefb0c99c7c0a367059f875c790d7928  Please note that the certificate-key gives access to cluster sensitive data, keep it secret! As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you can use "kubeadm init phase upload-certs --upload-certs" to reload certs afterward.  Then you can join any number of worker nodes by running the following on each as root:  kubeadm join 172.31.10.115:6443 --token vlkjp4.l71xfzz6woofctbv \  --discovery-token-ca-cert-hash sha256:4ec7dd568e841323c0f6c636045fc3e264f1562c95e6e5b5cd83fbffce2c0166 |

Tiếp theo chúng ta sẽ chạy câu lệnh sau, theo gợi ý của kubeadm init để sử kubectl cho user chạy ứng dụng

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

* Join các node master, worker theo kết quả ở bước khởi tạo cluster trên (trước thực hiện join vào cluster phải kiểm tra trạng thái của các kubelet ở các node)

1. ssh vào server master\_k8s\_02, master\_k8s\_03 chạy câu lệnh sau

|  |
| --- |
| kubeadm join 172.31.10.115:6443 --token vlkjp4.l71xfzz6woofctbv \  --discovery-token-ca-cert-hash sha256:4ec7dd568e841323c0f6c636045fc3e264f1562c95e6e5b5cd83fbffce2c0166 \  --control-plane --certificate-key 9c541d57ed688f9a65394455161e899eaefb0c99c7c0a367059f875c790d7928 |

1. ssh vào server worker\_k8s\_01, worker\_k8s\_02, worker\_k8s\_01, worker\_k8s\_03,worker\_k8s\_04, worker\_k8s\_05, worker\_k8s\_06 chạy câu lệnh sau

|  |
| --- |
| kubeadm join 172.31.10.115:6443 --token vlkjp4.l71xfzz6woofctbv \  --discovery-token-ca-cert-hash sha256:4ec7dd568e841323c0f6c636045fc3e264f1562c95e6e5b5cd83fbffce2c0166 |

### 

* Cài đặt giao diện mạng calico sử dụng bởi các Pod

Sau khi join các các node master, worker vào cluster thì các node ở trạng thái NotReady do chưa cài đặt giao diện mạng, thực hiện cài đặt như sau

|  |
| --- |
| **#** cd ~/k8s **#** kubectl apply -f calico.yaml  configmap/calico-config created  Warning: apiextensions.k8s.io/v1beta1 CustomResourceDefinition is deprecated in v1.16+, unavailable in v1.22+; use apiextensions.k8s.io/v1 CustomResourceDefinition  customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created  customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created  clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created  clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created  clusterrole.rbac.authorization.k8s.io/calico-node created  clusterrolebinding.rbac.authorization.k8s.io/calico-node created  daemonset.apps/calico-node created  serviceaccount/calico-node created  deployment.apps/calico-kube-controllers created  serviceaccount/calico-kube-controllers created |

# Cài đặt rancher

Sử dụng 1 node etcd để cài đặt rancher, sử dụng node etcd\_k8s\_01 (10.207.58.1) để thực hiện cài đặt rancher

## Download image để cài đặt rancher

Sử dụng docker để pull phiên bản mới nhất của rancher về và save lại thành file .tar như sau

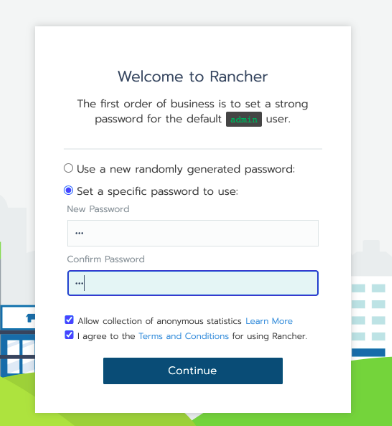
**$** docker pull rancher/rancher:latest  
**$** docker save rancher/rancher:latest > rancher.tar

Sau đó scp file rancher.tar lên server etcd\_k8s\_01 sử dụng docker load và chạy rancher như sau

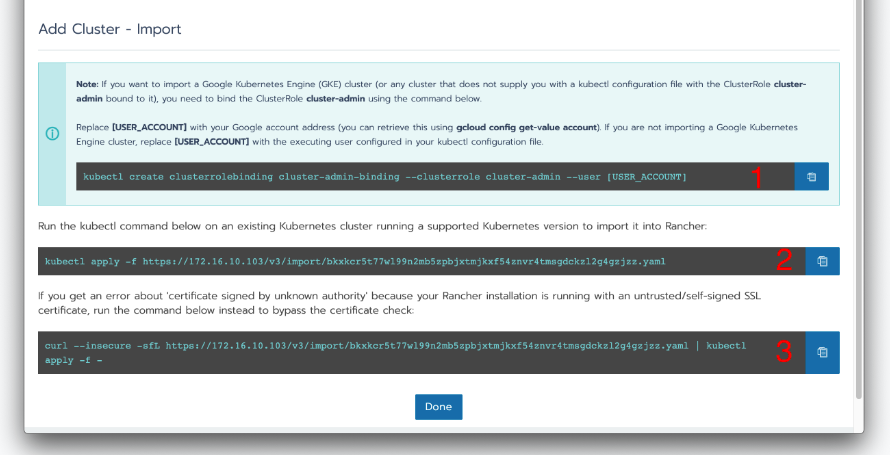
## Cài đặt racher

|  |
| --- |
| **$** docker load < rancher.tar **$** docker run -d --restart=unless-stopped -p 9001:80 -p 9000:443 rancher/rancher:latest |

* Sau khi triển khai xong truy cập vào địa chỉ https://10.207.58.1:9000 để thiết lập password cho user admin



* Sau khi thiết lập password, màn hình tiếp theo yêu cầu nhập địa chỉ url của server racher (etcd\_k8s\_01), điên ip 10.207.58.1
* Thao tác trên giao diện rancher thêm mới cluster chọn import an existing cluster đến màn hình cuối như sau



* Copy mã lệnh (3) vào một node master chạy để thực thi

# Cài đặt prometheus

## Download những images cần thiết

Sử dụng docker pull các images cần thiết từ máy local, sau đó save các images đó ra file .tar và scp lên các node worker

**#***# pull các image về máy local*  
**$** docker pull quay.io/coreos/kube-state-metrics:v1.9.7  
**$** docker pull quay.io/prometheus/node-exporter:v1.0.1  
**$** docker pull quay.io/prometheus/alertmanager:v0.21.0  
**$** docker pull jimmidyson/configmap-reload:v0.5.0  
**$** docker pull quay.io/prometheus/prometheus:v2.24.0  
**$** docker pull quay.io/prometheus/alertmanager:v0.21.0  
**#***# save image thành file .tar*  
**$** docker save quay.io/coreos/kube-state-metrics:v1.9.7 > kube-state-metrics.tar  
**$** docker save quay.io/prometheus/node-exporter:v1.0.1 > node-exporter.tar  
**$** docker save quay.io/prometheus/alertmanager:v0.21.0 > alertmanager.tar  
**$** docker save jimmidyson/configmap-reload:v0.5.0 > configmap-reload.tar  
**$** docker save quay.io/prometheus/prometheus:v2.24.0 > prometheus.tar  
**$** docker save quay.io/prometheus/alertmanager:v0.21.0 > alertmanager.tar

## Cài đặt [localpath](https://github.com/rancher/local-path-provisioner) làm storage cho prometheus server

1. Download images cần thiết để cài localpath storage và lưu thành file tar như sau

|  |
| --- |
| **$** docker pull rancher/local-path-provisioner:v0.0.19  **$** docker pull busybox  **$** docker save rancher/local-path-provisioner:v0.0.19 > local-path-provisioner.tar  **$** docker save busybox > busybox.tar |

1. Download file cấu hình để deploy localpath

|  |
| --- |
| wget <https://raw.githubusercontent.com/rancher/local-path-provisioner/master/deploy/local-path-storage.yaml> |

1. Sau đó scp images lên server các worker và scp file yaml trên lên 1 server master để deploy localpath
2. Load image và thực hiện deploy localpath như sau

|  |
| --- |
| **#** Ở các worker **$** docker load < local-path-provisioner.tar **$** docker load < busybox.tar **#** deploy ở 1 node master chưa file yaml **$** kubectl apply -f local-path-storage.yaml |

## Tạo các file yaml để deploy prometheus

* Tạo namespace

|  |
| --- |
| apiVersion: v1 kind: Namespace metadata:  name: metrics |

* Tạo serviceaccount

1. Acc cho kube-state-metrics giao tiếp với kubernetes api

|  |
| --- |
| apiVersion: v1 kind: ServiceAccount metadata:  labels:  app.kubernetes.io/name: kube-state-metrics  helm.sh/chart: kube-state-metrics-2.9.8  app.kubernetes.io/managed-by: Helm  app.kubernetes.io/instance: release-name  name: kube-state-metrics  namespace: metrics |

1. Acc cho alertmanager giao tiếp với kubernetes api

|  |
| --- |
| apiVersion: v1 kind: ServiceAccount metadata:  labels:  component: "alertmanager"  app: prometheus  release: release-name  chart: prometheus-13.3.3  heritage: Helm  name: prometheus-alertmanager  namespace: metrics |

1. Acc cho node exporter giao tiếp với kubernetes api

|  |
| --- |
| apiVersion: v1 kind: ServiceAccount metadata:  labels:  component: "node-exporter"  app: prometheus  release: release-name  chart: prometheus-13.3.3  heritage: Helm  name: prometheus-node-exporter  namespace: metrics |

1. Acc cho prometheus server giao tiếp với kubernetes api

|  |
| --- |
| apiVersion: v1 kind: ServiceAccount metadata:  labels:  component: "server"  app: prometheus  release: release-name  chart: prometheus-13.3.3  heritage: Helm  name: prometheus-server  namespace: metrics |

* Cấu hình ClusterRole cho kube-state-metrics để có thể truy cập vào kubernetes cluster đọc các dữ liệu metrics.

|  |
| --- |
| apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata:  labels:  app.kubernetes.io/name: kube-state-metrics  helm.sh/chart: kube-state-metrics-2.9.8  app.kubernetes.io/managed-by: Helm  app.kubernetes.io/instance: release-name  name: kube-state-metrics rules: - apiGroups: ["certificates.k8s.io"]  resources:  - certificatesigningrequests  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - configmaps  verbs: ["list", "watch"] - apiGroups: ["batch"]  resources:  - cronjobs  verbs: ["list", "watch"] - apiGroups: ["extensions", "apps"]  resources:  - daemonsets  verbs: ["list", "watch"] - apiGroups: ["extensions", "apps"]  resources:  - deployments  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - endpoints  verbs: ["list", "watch"] - apiGroups: ["autoscaling"]  resources:  - horizontalpodautoscalers  verbs: ["list", "watch"] - apiGroups: ["extensions", "networking.k8s.io"]  resources:  - ingresses  verbs: ["list", "watch"] - apiGroups: ["batch"]  resources:  - jobs  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - limitranges  verbs: ["list", "watch"] - apiGroups: ["admissionregistration.k8s.io"]  resources:  - mutatingwebhookconfigurations  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - namespaces  verbs: ["list", "watch"] - apiGroups: ["networking.k8s.io"]  resources:  - networkpolicies  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - nodes  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - persistentvolumeclaims  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - persistentvolumes  verbs: ["list", "watch"] - apiGroups: ["policy"]  resources:  - poddisruptionbudgets  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - pods  verbs: ["list", "watch"] - apiGroups: ["extensions", "apps"]  resources:  - replicasets  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - replicationcontrollers  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - resourcequotas  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - secrets  verbs: ["list", "watch"] - apiGroups: [""]  resources:  - services  verbs: ["list", "watch"] - apiGroups: ["apps"]  resources:  - statefulsets  verbs: ["list", "watch"] - apiGroups: ["storage.k8s.io"]  resources:  - storageclasses  verbs: ["list", "watch"] - apiGroups: ["admissionregistration.k8s.io"]  resources:  - validatingwebhookconfigurations  verbs: ["list", "watch"] - apiGroups: ["storage.k8s.io"]  resources:  - volumeattachments  verbs: ["list", "watch"] |

* Cấu hình ClusterRole cho prometheus server

|  |
| --- |
| apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata:  labels:  component: "server"  app: prometheus  release: release-name  chart: prometheus-13.3.3  heritage: Helm  name: prometheus-server rules:  - apiGroups:  - ""  resources:  - nodes  - nodes/proxy  - nodes/metrics  - services  - endpoints  - pods  - ingresses  - configmaps  verbs:  - get  - list  - watch  - apiGroups:  - "extensions"  - "networking.k8s.io"  resources:  - ingresses/status  - ingresses  verbs:  - get  - list  - watch  - nonResourceURLs:  - "/metrics"  verbs:  - get |

* Cấu hình ClusterRole cho alertmanager

|  |
| --- |
| apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata:  labels:  component: "alertmanager"  app: prometheus  release: release-name  chart: prometheus-13.3.3  heritage: Helm  name: prometheus-alertmanager rules:  [] |

## Triển khai prometheus

Download file [prometheus.yaml](https://raw.githubusercontent.com/cuongnb/k8s-checklist/master/deployment/metrics/prometheus.yaml) từ máy local về và scp lên một node master và thực hiện câu lệnh sau

|  |
| --- |
| **$** kubectl apply -f prometheus.yaml |

# Cài đặt grafana

## Download những images cần thiết

Sử dụng docker pull các images để cài đặt grafana từ máy local sau đó save lại thành file .tar, scp toàn bộ các file .tar và thực hiện load lại trên các worker

**#***# pull các image về máy local*  
**$** docker pull curlimages/curl:7.73.0  
**$** docker pull grafana/grafana:8.0.1  
**$** docker pull bats/bats:v1.1.0  
**#***# save image thành file .tar*  
**$** docker save curlimages/curl:7.73.0 > curl.tar  
**$** docker save grafana/grafana:8.0.1 > grafana.tar  
**$** docker save bats/bats:v1.1.0 > bats.tar

* Scp các file jar lên các node worker và thực hiện load vào docker như sau

**$** docker load < curl.tar  
**$** docker load < grafana.tar  
**$** docker load < bats.tar

## Triển khai grafana

Download file [grafana.yaml](https://raw.githubusercontent.com/cuongnb/k8s-checklist/master/deployment/metrics/grafana.yaml) và file [grafana-dashboard.yaml](https://raw.githubusercontent.com/cuongnb/k8s-checklist/master/deployment/metrics/grafana-dashboard.yaml) từ máy local và scp lên một node master và thực hiện câu lệnh sau

**$** kubectl apply -f grafana.yaml,grafana-dashboard.yaml

# Triển khai log tập trung

Phần này sử dụng fluentd để đẩy log vào elasticsearch, và sử dụng kibana để visualize liệu thu thập được. Elasticsearch được cài trên cụm (10.207.58.13, 10.207.58.14, 10.207.58.13) với port 9200

## Download images cần thiết.

|  |
| --- |
| **$** docker pull k8s.gcr.io/fluentd-elasticsearch:v2.4.0 **$** docker.elastic.co/kibana/kibana-oss:6.8.16 **$** docker save k8s.gcr.io/fluentd-elasticsearch:v2.4.0 > fluentd-elasticsearch.tar **$** docker save kibana-oss:6.8.16 > kibana-oss.tar |

## Cấu hình file yaml

* Tạo file logging-namespace.yaml như sau

|  |
| --- |
| kind: Namespace apiVersion: v1 metadata: name: logging labels:  k8s-app: logging  kubernetes.io/cluster-service: "true"  addonmanager.kubernetes.io/mode: Reconcile |

Sau đó thực hiện lệnh kubectl apply -f logging-namespace.yaml để tạo namespace cho logging

* Tạo file fluentd-es-configmap.yaml cấu hình output trỏ về elasticsearch như sau

|  |
| --- |
| kind: ConfigMap apiVersion: v1 metadata:  name: fluentd-es-config-v0.2.1  namespace: logging  labels:  addonmanager.kubernetes.io/mode: Reconcile data:  system.conf: |-  <system>  root\_dir /tmp/fluentd-buffers/  </system>  containers.input.conf: |-  system.input.conf: |-  forward.input.conf: |-  monitoring.conf: |-  output.conf: |-  <match \*\*>  @id elasticsearch  @type elasticsearch  @log\_level info  type\_name \_doc  include\_tag\_key true  host 10.207.58.14  port 9200  logstash\_format true  <buffer>  @type file  path /var/log/fluentd-buffers/kubernetes.system.buffer  flush\_mode interval  retry\_type exponential\_backoff  flush\_thread\_count 2  flush\_interval 5s  retry\_forever  retry\_max\_interval 30  chunk\_limit\_size 2M  total\_limit\_size 500M  overflow\_action block  </buffer>  </match> |

* Tạo file fluentd-es-ds,yaml nhu sau

1. Tạo ServiceAccount có tên là fluentd-es để truy cập vào Kubernetes API

|  |
| --- |
| apiVersion: v1 kind: ServiceAccount metadata:  name: fluentd-es  namespace: logging  labels:  k8s-app: fluentd-es  kubernetes.io/cluster-service: "true"  addonmanager.kubernetes.io/mode: Reconcile |

1. Định nghĩa ClusterRole có tên là fluentd-es và cấp quyền get, list, watch trên các pod và namespace

|  |
| --- |
| kind: ClusterRole apiVersion: rbac.authorization.k8s.io/v1 metadata:  name: fluentd-es  labels:  k8s-app: fluentd-es  kubernetes.io/cluster-service: "true"  addonmanager.kubernetes.io/mode: Reconcile rules: - apiGroups:  - ""  resources:  - "namespaces"  - "pods"  verbs:  - "get"  - "watch"  - "list" |

1. Tạo ClusterRoleBinding để gán ClusterRole (fluentd-es) cho ServiceAccount (fluentd-es)

|  |
| --- |
| kind: ClusterRoleBinding apiVersion: rbac.authorization.k8s.io/v1 metadata:  name: fluentd-es  labels:  k8s-app: fluentd-es  kubernetes.io/cluster-service: "true"  addonmanager.kubernetes.io/mode: Reconcile subjects: - kind: ServiceAccount  name: fluentd-es  namespace: logging  apiGroup: "" roleRef:  kind: ClusterRole  name: fluentd-es  apiGroup: "" |

1. Tạo DaemonSet như sau

|  |
| --- |
| apiVersion: apps/v1 kind: DaemonSet metadata:  name: fluentd-es-v2.4.0  namespace: logging  labels:  k8s-app: fluentd-es  version: v2.4.0  kubernetes.io/cluster-service: "true"  addonmanager.kubernetes.io/mode: Reconcile spec:  selector:  matchLabels:  k8s-app: fluentd-es  version: v2.4.0  template:  metadata:  labels:  k8s-app: fluentd-es  kubernetes.io/cluster-service: "true"  version: v2.4.0  annotations:  scheduler.alpha.kubernetes.io/critical-pod: ''  seccomp.security.alpha.kubernetes.io/pod: 'docker/default'  spec:  priorityClassName: system-node-critical  serviceAccountName: fluentd-es  containers:  - name: fluentd-es  image: k8s.gcr.io/fluentd-elasticsearch:v2.4.0  env:  - name: FLUENTD\_ARGS  value: --no-supervisor -q  resources:  limits:  memory: 500Mi  requests:  cpu: 100m  memory: 200Mi  volumeMounts:  - name: varlog  mountPath: /var/log  - name: varlibdockercontainers  mountPath: /var/lib/docker/containers  readOnly: true  - name: config-volume  mountPath: /etc/fluent/config.d  nodeSelector:  beta.kubernetes.io/fluentd-ds-ready: "true"  terminationGracePeriodSeconds: 30  volumes:  - name: varlog  hostPath:  path: /var/log  - name: varlibdockercontainers  hostPath:  path: /var/lib/docker/containers  - name: config-volume  configMap:  name: fluentd-es-config-v0.2.0 |

* Tạo file kibana-deployment.yaml như sau

|  |
| --- |
| apiVersion: apps/v1 kind: Deployment metadata:  name: kibana-logging  namespace: logging  labels:  k8s-app: kibana-logging  kubernetes.io/cluster-service: "true"  addonmanager.kubernetes.io/mode: Reconcile spec:  replicas: 1  selector:  matchLabels:  k8s-app: kibana-logging  template:  metadata:  labels:  k8s-app: kibana-logging  annotations:  seccomp.security.alpha.kubernetes.io/pod: 'docker/default'  spec:  containers:  - name: kibana-logging  image: docker.elastic.co/kibana/kibana-oss:6.8.16  resources:  # need more cpu upon initialization, therefore burstable class  limits:  cpu: 1000m  requests:  cpu: 100m  env:  - name: ELASTICSEARCH\_URL  value: http://10.207.58.14:9200  ports:  - containerPort: 5601  name: ui  protocol: TCP |

* Tạo file kibana-service.yaml như sau

|  |
| --- |
| apiVersion: v1 kind: Service metadata:  name: kibana-logging  namespace: logging  labels:  k8s-app: kibana-logging  kubernetes.io/cluster-service: "true"  addonmanager.kubernetes.io/mode: Reconcile  kubernetes.io/name: "Kibana" spec:  ports:  - port: 5601  protocol: TCP  targetPort: ui  nodePort: 30561  selector:  k8s-app: kibana-logging  type: NodePort |

Thực hiện deploy các file yaml ở trên như sau

|  |
| --- |
| kubectl apply -f logging-namespace.yaml kubectl apply -f fluentd-es-configmap.yaml kubectl apply -f fluentd-es-ds.yaml kubectl apply -f kibana-deployment.yaml kubectl apply -f kibana-service.yaml |