CRI-O기반의 k8s 설치

사전사항

1. OS환경설정

```
$> swapoff -a

$> 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
```

2. crio / kubernetees 패키지 리포지터리 구성

```
$> cat /etc/yum.repos.d/libcontainers.repo
[devel_kubic_libcontainers_stable]
name=Stable Releases of Upstream github.com/containers packages (CentOS_8)
type=rpm-md
baseurl=https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/CentOS_8/
gpgcheck=1
gpgkey=https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/CentOS_8/repodata/repomd.xml.key
enabled=1
```

```
$> cat /etc/yum.repos.d/cri-o-1.23.repo
[devel_kubic_libcontainers_stable_cri-o_1.23]
name=devel:kubic:libcontainers:stable:cri-o:1.23 (CentOS_8)
type=rpm-md
baseurl=https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/1.23/CentOS_8/
gpgcheck=1
gpgkey=https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/1.23/CentOS_8/repodata/repomd.xml.key
enabled=1
```

```
$> cat /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-$basearch
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
```

3. 패키지 설치

```
$> yum install kubelet kubeadm kubectl libcgroup cri-o cri-tools -y
$> systemctl enable crio --now
$> systemctl enable kubelet
```

클러스터 생성 (control palin 1번에서만 수행)

1. kubeadm 클러스터 생성

```
$> kubeadm init --control-plane-endpoint 172.21.107.238:6443 --pod-network-cidr 10.250.0.0/16 --ignore-preflight-errors=all --upload-certs

## 결과값중에 control / worker 노드별 join 명령이 다르기 때문에 별도로 복사해놓어야 함
```

##Control node용

```
$> kubeadm join 172.21.107.238:6443 --token abcd \
      --discovery-token-ca-cert-hash sha256:yyy \
      --control-plane --certificate-key zzz
```

Worker Node용

```
$> kubeadm join 172.21.107.238:6443 --token abcd \
      --discovery-token-ca-cert-hash sha256:yyyy \
 --ignore-preflight-errors=all
```

2. 인증서 정보 복사

```
$> mkdir -p $HOME/.kube
$> /bin/cp /etc/kubernetes/admin.conf $HOME/.kube/config
$> chown $(id -u):$(id -g) $HOME/.kube/config
$> export KUBECONFIG=/etc/kubernetes/admin.conf
```

3. CNI 설치(Calico)

```
$> curl https://projectcalico.docs.tigera.io/manifests/calico.yaml -O
$> kubectl apply -f calico.yaml
```

클러스터 연동

1. 타 Control plain 연동 (Control Plain 한대씩 순차 작업 수행)

```
$> kubeadm join 172.21.107.238:6443 --token abcd \
      --discovery-token-ca-cert-hash sha256:yyy \
      --control-plane --certificate-key zzz
```

2. 노드 연동 확인 (Control plain에서 수행)

```
$> kubectl get no
NAME                    STATUS   ROLES                  AGE    VERSION
k8stesttx-k8s-master-dev01   Ready    control-plane,master   3h6m   v1.23.5
k8stesttx-k8s-master-dev02   Ready    control-plane,master   3h6m   v1.23.5
k8stesttx-k8s-master-dev03   Ready    control-plane,master   3h6m   v1.23.5
```

3. Worker Node 연동

```
$> kubeadm join 172.21.107.238:6443 --token abcd \
      --discovery-token-ca-cert-hash sha256:yyyy \
 --ignore-preflight-errors=all
```

4. 노드 연동 확인 (Control plain에서 수행)

```
$> kubectl get no
NAME                    STATUS   ROLES                  AGE    VERSION
...
k8stesttx-k8s-worker-dev01   Ready    <none>                 40m    v1.23.5
k8stesttx-k8s-worker-dev02   Ready    <none>                 40m    v1.23.5
k8stesttx-k8s-worker-dev03   Ready    <none>                 40m    v1.23.5
```

k8s 인증서 10년으로 연장

1. 인증서 연장 스크립트

```
#!/usr/bin/env bash

set -o errexit
set -o pipefail
# set -o xtrace

# set output color
NC='\033[0m'
RED='\033[31m'
GREEN='\033[32m'
YELLOW='\033[33m'
BLUE='\033[34m'

log::err() {
 printf "[$(date +'%Y-%m-%dT%H:%M:%S.%2N%z')][${RED}ERROR${NC}] %b\n" "$@"
}

log::info() {
 printf "[$(date +'%Y-%m-%dT%H:%M:%S.%2N%z')][INFO] %b\n" "$@"
}

log::warning() {
 printf "[$(date +'%Y-%m-%dT%H:%M:%S.%2N%z')][${YELLOW}WARNING${NC}] \033[0m%b\n" "$@"
}

check_file() {
 if [[ ! -r ${1} ]]; then
    log::err "can not find ${1}"
   exit 1
 fi
}

# get x509v3 subject alternative name from the old certificate
cert::get_subject_alt_name() {
 local cert=${1}.crt
 local alt_name

  check_file "${cert}"
 alt_name=$(openssl x509 -text -noout -in "${cert}" | grep -A1 'Alternative' | tail -n1 | sed 's/[[:space:]]*Address//g')
 printf "%s\n" "${alt_name}"
}

# get subject from the old certificate
cert::get_subj() {
 local cert=${1}.crt
 local subj

  check_file "${cert}"
 subj=$(openssl x509 -text -noout -in "${cert}" | grep "Subject:" | sed 's/Subject:/\//g;s/\,/\//;s/[[:space:]]//g')
 printf "%s\n" "${subj}"
}

cert::backup_file() {
 local file=${1}
 if [[ ! -e ${file}.old-$(date +%Y%m%d) ]]; then
    cp -rp "${file}" "${file}.old-$(date +%Y%m%d)"
    log::info "backup ${file} to ${file}.old-$(date +%Y%m%d)"
 else
    log::warning "does not backup, ${file}.old-$(date +%Y%m%d) already exists"
 fi
}

# check certificate expiration
cert::check_cert_expiration() {
 local cert=${1}.crt
 local cert_expires
```

```
 cert_expires=$(openssl x509 -text -noout -in "${cert}" | awk -F ": " '/Not After/{print$2}')
 printf "%s\n" "${cert_expires}"
}

# check kubeconfig expiration
cert::check_kubeconfig_expiration() {
 local config=${1}.conf
 local cert
 local cert_expires

 cert=$(grep "client-certificate-data" "${config}" | awk '{print$2}' | base64 -d)
 cert_expires=$(openssl x509 -text -noout -in <(printf "%s" "${cert}") | awk -F ": " '/Not After/{print$2}')
 printf "%s\n" "${cert_expires}"
}

# check etcd certificates expiration
cert::check_etcd_certs_expiration() {
 local cert
 local certs

 certs=(
   "${ETCD_CERT_CA}"
   "${ETCD_CERT_SERVER}"
   "${ETCD_CERT_PEER}"
   "${ETCD_CERT_HEALTHCHECK_CLIENT}"
   "${ETCD_CERT_APISERVER_ETCD_CLIENT}"
 )

 for cert in "${certs[@]}"; do
   if [[ ! -r ${cert} ]]; then
     printf "%-50s%-30s\n" "${cert}.crt" "$(cert::check_cert_expiration "${cert}")"
   fi
 done
}

# check master certificates expiration
cert::check_master_certs_expiration() {
 local certs
 local kubeconfs
 local cert
 local conf

 certs=(
   "${CERT_CA}"
   "${CERT_APISERVER}"
   "${CERT_APISERVER_KUBELET_CLIENT}"
   "${FRONT_PROXY_CA}"
   "${FRONT_PROXY_CLIENT}"
 )

 kubeconfs=(
   "${CONF_CONTROLLER_MANAGER}"
   "${CONF_SCHEDULER}"
   "${CONF_ADMIN}"
 )

 printf "%-50s%-30s\n" "CERTIFICATE" "EXPIRES"

 for conf in "${kubeconfs[@]}"; do
   if [[ ! -r ${conf} ]]; then
     printf "%-50s%-30s\n" "${conf}.config" "$(cert::check_kubeconfig_expiration "${conf}")"
   fi
 done

 for cert in "${certs[@]}"; do
   if [[ ! -r ${cert} ]]; then
     printf "%-50s%-30s\n" "${cert}.crt" "$(cert::check_cert_expiration "${cert}")"
   fi
 done
```

```
}

# check all certificates expiration
cert::check_all_expiration() {
  cert::check_master_certs_expiration
  cert::check_etcd_certs_expiration
}

# generate certificate whit client, server or peer
# Args:
#   $1 (the name of certificate)
#   $2 (the type of certificate, must be one of client, server, peer)
#   $3 (the subject of certificates)
#   $4 (the validity of certificates) (days)
#   $5 (the name of ca)
#   $6 (the x509v3 subject alternative name of certificate when the type of certificate is server or peer)
cert::gen_cert() {
 local cert_name=${1}
 local cert_type=${2}
 local subj=${3}
 local cert_days=${4}
 local ca_name=${5}
 local alt_name=${6}
 local ca_cert=${ca_name}.crt
 local ca_key=${ca_name}.key
 local cert=${cert_name}.crt
 local key=${cert_name}.key
 local csr=${cert_name}.csr
 local common_csr_conf='distinguished_name = dn\n[dn]\n[v3_ext]\nkeyUsage = critical, digitalSignature, keyEncipherment\n'

 for file in "${ca_cert}" "${ca_key}" "${cert}" "${key}"; do
    check_file "${file}"
 done

 case "${cert_type}" in
  client)
   csr_conf=$(printf "%bextendedKeyUsage = clientAuth\n" "${common_csr_conf}")
    ;;
  server)
   csr_conf=$(printf "%bextendedKeyUsage = serverAuth\nsubjectAltName = %b\n" "${common_csr_conf}" "${alt_name}")
    ;;
  peer)
   csr_conf=$(printf "%bextendedKeyUsage = serverAuth, clientAuth\nsubjectAltName = %b\n" "${common_csr_conf}" "${alt_name}")
    ;;
  *)
    log::err "unknow, unsupported certs type: ${YELLOW}${cert_type}${NC}, supported type: client, server, peer"
   exit 1
    ;;
 esac

 # gen csr
 openssl req -new -key "${key}" -subj "${subj}" -reqexts v3_ext \
   -config <(printf "%b" "${csr_conf}") \
   -out "${csr}" >/dev/null 2>&1
 # gen cert
 openssl x509 -in "${csr}" -req -CA "${ca_cert}" -CAkey "${ca_key}" -CAcreateserial -extensions v3_ext \
   -extfile <(printf "%b" "${csr_conf}") \
   -days "${cert_days}" -out "${cert}" >/dev/null 2>&1

  rm -f "${csr}"
}

cert::update_kubeconf() {
 local cert_name=${1}
 local kubeconf_file=${cert_name}.conf
 local cert=${cert_name}.crt
 local key=${cert_name}.key
 local subj
```

```
  local cert_base64

   check_file "${kubeconf_file}"
  # get the key from the old kubeconf
  grep "client-key-data" "${kubeconf_file}" | awk '{print$2}' | base64 -d >"${key}"
  # get the old certificate from the old kubeconf
  grep "client-certificate-data" "${kubeconf_file}" | awk '{print$2}' | base64 -d >"${cert}"
  # get subject from the old certificate
  subj=$(cert::get_subj "${cert_name}")
   cert::gen_cert "${cert_name}" "client" "${subj}" "${CERT_DAYS}" "${CERT_CA}"
  # get certificate base64 code
  cert_base64=$(base64 -w 0 "${cert}")

  # set certificate base64 code to kubeconf
  sed -i 's/client-certificate-data:.*/client-certificate-data: "'"${cert_base64}"'"/g' "${kubeconf_file}"

   rm -f "${cert}"
   rm -f "${key}"
 }

 cert::update_etcd_cert() {
  local subj
  local subject_alt_name
  local cert

  # generate etcd server,peer certificate
  # /etc/kubernetes/pki/etcd/server
  # /etc/kubernetes/pki/etcd/peer
  for cert in ${ETCD_CERT_SERVER} ${ETCD_CERT_PEER}; do
    subj=$(cert::get_subj "${cert}")
    subject_alt_name=$(cert::get_subject_alt_name "${cert}")
     cert::gen_cert "${cert}" "peer" "${subj}" "${CERT_DAYS}" "${ETCD_CERT_CA}" "${subject_alt_name}"
     log::info "${GREEN}updated ${BLUE}${cert}.conf${NC}"
  done

  # generate etcd healthcheck-client,apiserver-etcd-client certificate
  # /etc/kubernetes/pki/etcd/healthcheck-client
  # /etc/kubernetes/pki/apiserver-etcd-client
  for cert in ${ETCD_CERT_HEALTHCHECK_CLIENT} ${ETCD_CERT_APISERVER_ETCD_CLIENT}; do
    subj=$(cert::get_subj "${cert}")
     cert::gen_cert "${cert}" "client" "${subj}" "${CERT_DAYS}" "${ETCD_CERT_CA}"
     log::info "${GREEN}updated ${BLUE}${cert}.conf${NC}"
  done

  # restart etcd
  docker ps | awk '/k8s_etcd/{print$1}' | xargs -r -I '{}' docker restart {} >/dev/null 2>&1 || true
   log::info "restarted etcd"
 }

 cert::update_master_cert() {
  local subj
  local subject_alt_name
  local conf

  # generate apiserver server certificate
  # /etc/kubernetes/pki/apiserver
  subj=$(cert::get_subj "${CERT_APISERVER}")
  subject_alt_name=$(cert::get_subject_alt_name "${CERT_APISERVER}")
   cert::gen_cert "${CERT_APISERVER}" "server" "${subj}" "${CERT_DAYS}" "${CERT_CA}" "${subject_alt_name}"
   log::info "${GREEN}updated ${BLUE}${CERT_APISERVER}.crt${NC}"

  # generate apiserver-kubelet-client certificate
  # /etc/kubernetes/pki/apiserver-kubelet-client
  subj=$(cert::get_subj "${CERT_APISERVER_KUBELET_CLIENT}")
   cert::gen_cert "${CERT_APISERVER_KUBELET_CLIENT}" "client" "${subj}" "${CERT_DAYS}" "${CERT_CA}"
   log::info "${GREEN}updated ${BLUE}${CERT_APISERVER_KUBELET_CLIENT}.crt${NC}"

  # generate kubeconf for controller-manager,scheduler and kubelet
  # /etc/kubernetes/controller-manager,scheduler,admin,kubelet.conf
```

```
  for conf in ${CONF_CONTROLLER_MANAGER} ${CONF_SCHEDULER} ${CONF_ADMIN} ${CONF_KUBELET}; do
    if [[ ${conf##*/} == "kubelet" ]]; then
      # https://github.com/kubernetes/kubeadm/issues/1753
      set +e
       grep kubelet-client-current.pem /etc/kubernetes/kubelet.conf >/dev/null 2>&1
      kubelet_cert_auto_update=$?
      set -e
      if [[ "$kubelet_cert_auto_update" == "0" ]]; then
         log::info "does not need to update kubelet.conf"
        continue
      fi
    fi

    # update kubeconf
    cert::update_kubeconf "${conf}"
     log::info "${GREEN}updated ${BLUE}${conf}.conf${NC}"

    # copy admin.conf to ${HOME}/.kube/config
    if [[ ${conf##*/} == "admin" ]]; then
       mkdir -p "${HOME}/.kube"
      local config=${HOME}/.kube/config
      local config_backup
      config_backup=${HOME}/.kube/config.old-$(date +%Y%m%d)
      if [[ -f ${config} ]] && [[ ! -f ${config_backup} ]]; then
         cp -fp "${config}" "${config_backup}"
         log::info "backup ${config} to ${config_backup}"
      fi
       cp -fp "${conf}.conf" "${HOME}/.kube/config"
       log::info "copy the admin.conf to ${HOME}/.kube/config"
    fi
  done

  # generate front-proxy-client certificate
  # /etc/kubernetes/pki/front-proxy-client
  subj=$(cert::get_subj "${FRONT_PROXY_CLIENT}")
   cert::gen_cert "${FRONT_PROXY_CLIENT}" "client" "${subj}" "${CERT_DAYS}" "${FRONT_PROXY_CA}"
   log::info "${GREEN}updated ${BLUE}${FRONT_PROXY_CLIENT}.crt${NC}"

  # restart apiserver, controller-manager, scheduler and kubelet
  for item in "apiserver" "controller-manager" "scheduler"; do
     docker ps | awk '/k8s_kube-'${item}'/{print$1}' | xargs -r -I '{}' docker restart {} >/dev/null 2>&1 || true
     log::info "restarted ${item}"
  done
   systemctl restart kubelet || true
   log::info "restarted kubelet"
}

main() {
  local node_type=$1

  CERT_DAYS=3650

  KUBE_PATH=/etc/kubernetes
  PKI_PATH=${KUBE_PATH}/pki

  # master certificates path
  # apiserver
  CERT_CA=${PKI_PATH}/ca
  CERT_APISERVER=${PKI_PATH}/apiserver
  CERT_APISERVER_KUBELET_CLIENT=${PKI_PATH}/apiserver-kubelet-client
  CONF_CONTROLLER_MANAGER=${KUBE_PATH}/controller-manager
  CONF_SCHEDULER=${KUBE_PATH}/scheduler
  CONF_ADMIN=${KUBE_PATH}/admin
  CONF_KUBELET=${KUBE_PATH}/kubelet
  # front-proxy
  FRONT_PROXY_CA=${PKI_PATH}/front-proxy-ca
  FRONT_PROXY_CLIENT=${PKI_PATH}/front-proxy-client
```

```
# etcd certificates path
ETCD_CERT_CA=${PKI_PATH}/etcd/ca
ETCD_CERT_SERVER=${PKI_PATH}/etcd/server
ETCD_CERT_PEER=${PKI_PATH}/etcd/peer
ETCD_CERT_HEALTHCHECK_CLIENT=${PKI_PATH}/etcd/healthcheck-client
ETCD_CERT_APISERVER_ETCD_CLIENT=${PKI_PATH}/apiserver-etcd-client

case ${node_type} in
# etcd)
# # update etcd certificates
#   cert::update_etcd_cert
# ;;
master)
  # check certificates expiration
  cert::check_master_certs_expiration
  # backup $KUBE_PATH to $KUBE_PATH.old-$(date +%Y%m%d)
  cert::backup_file "${KUBE_PATH}"
  # update master certificates and kubeconf
  log::info "${GREEN}updating...${NC}"
   cert::update_master_cert
   log::info "${GREEN}done!!!${NC}"
  # check certificates expiration after certificates updated
  cert::check_master_certs_expiration
   ;;
 all)
  # check certificates expiration
  cert::check_all_expiration
  # backup $KUBE_PATH to $KUBE_PATH.old-$(date +%Y%m%d)
  cert::backup_file "${KUBE_PATH}"
  # update etcd certificates
  log::info "${GREEN}updating...${NC}"
   cert::update_etcd_cert
  # update master certificates and kubeconf
  cert::update_master_cert
   log::info "${GREEN}done!!!${NC}"
  # check certificates expiration after certificates updated
  cert::check_all_expiration
   ;;
 check)
  # check certificates expiration
  cert::check_all_expiration
   ;;
 *)
   log::err "unknown, unsupported cert type: ${node_type}, supported type: \"all\", \"master\""
  printf "Documentation: https://github.com/yuyicai/update-kube-cert
 example:
   '\033[32m./update-kubeadm-cert.sh all\033[0m' update all etcd certificates, master certificates and kubeconf
     /etc/kubernetes
     ├── admin.conf
     ├── controller-manager.conf
     ├── scheduler.conf
     ├── kubelet.conf
     └── pki
         ├── apiserver.crt
         ├── apiserver-etcd-client.crt
         ├── apiserver-kubelet-client.crt
         ├── front-proxy-client.crt
         └── etcd
             ├── healthcheck-client.crt
             ├── peer.crt
             └── server.crt

   '\033[32m./update-kubeadm-cert.sh master\033[0m' update only master certificates and kubeconf
     /etc/kubernetes
     ├── admin.conf
     ├── controller-manager.conf
     ├── scheduler.conf
     ├── kubelet.conf
     └── pki
```

```
        ├── apiserver.crt
        ├── apiserver-kubelet-client.crt
        └── front-proxy-client.crt
"
  exit 1
   ;;
 esac
}

main "$@"
```

2. 업데이트 전 인증서 정보 확인

```
$> kubeadm certs check-expiration
[check-expiration] Reading configuration from the cluster...
[check-expiration] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'

CERTIFICATE                EXPIRES                  RESIDUAL TIME   CERTIFICATE AUTHORITY   EXTERNALLY MANAGED
admin.conf                 Apr 17, 2023 06:09 UTC   364d            ca                      no
apiserver                  Apr 17, 2023 06:09 UTC   364d            ca                      no
apiserver-etcd-client      Apr 17, 2023 06:09 UTC   364d            etcd-ca                 no
apiserver-kubelet-client   Apr 17, 2023 06:09 UTC   364d            ca                      no
controller-manager.conf    Apr 17, 2023 06:09 UTC   364d            ca                      no
etcd-healthcheck-client    Apr 17, 2023 06:09 UTC   364d            etcd-ca                 no
etcd-peer                  Apr 17, 2023 06:09 UTC   364d            etcd-ca                 no
etcd-server                Apr 17, 2023 06:09 UTC   364d            etcd-ca                 no
front-proxy-client         Apr 17, 2023 06:09 UTC   364d            front-proxy-ca          no
scheduler.conf             Apr 17, 2023 06:09 UTC   364d            ca                      no

CERTIFICATE AUTHORITY   EXPIRES                  RESIDUAL TIME   EXTERNALLY MANAGED
ca                      Apr 17, 2032 04:17 UTC   9y              no
etcd-ca                 Apr 17, 2032 04:17 UTC   9y              no
front-proxy-ca          Apr 17, 2032 04:17 UTC   9y              no
```

3. 인증서 업데이트 (Control Plain 1대씩 순차 작업 수행, 서버단위로 30초 가량 대기 필요)

```
$> chmod +x cert_update.sh
$> ./cert_update.sh
...
```

4. 인증서 갱신정보 확인

```
$> kubeadm certs check-expiration
[check-expiration] Reading configuration from the cluster...
[check-expiration] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'

CERTIFICATE                EXPIRES                  RESIDUAL TIME   CERTIFICATE AUTHORITY   EXTERNALLY MANAGED
admin.conf                 Apr 17, 2032 06:09 UTC   9y              ca                      no
apiserver                  Apr 17, 2032 06:09 UTC   9y              ca                      no
apiserver-etcd-client      Apr 17, 2032 06:09 UTC   9y              etcd-ca                 no
apiserver-kubelet-client   Apr 17, 2032 06:09 UTC   9y              ca                      no
controller-manager.conf    Apr 17, 2032 06:09 UTC   9y              ca                      no
etcd-healthcheck-client    Apr 17, 2032 06:09 UTC   9y              etcd-ca                 no
etcd-peer                  Apr 17, 2032 06:09 UTC   9y              etcd-ca                 no
etcd-server                Apr 17, 2032 06:09 UTC   9y              etcd-ca                 no
front-proxy-client         Apr 17, 2032 06:09 UTC   9y              front-proxy-ca          no
scheduler.conf             Apr 17, 2032 06:09 UTC   9y              ca                      no

CERTIFICATE AUTHORITY   EXPIRES                  RESIDUAL TIME   EXTERNALLY MANAGED
ca                      Apr 17, 2032 04:17 UTC   9y              no
etcd-ca                 Apr 17, 2032 04:17 UTC   9y              no
front-proxy-ca          Apr 17, 2032 04:17 UTC   9y              no
```

Reference

- *https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/*
- *https://github.com/yuyicai/update-kube-cert.git*

Revision #1
Created 8 June 2022 03:08:07 by artop0420
Updated 5 July 2023 00:26:12 by artop0420