Security (S7)

Kubernetes Security Primitives

- Hosts should be secured (no root access, pw based authentication disabled, only ssh)
- Kube-apiserver should be secured with authorization
    - Everything using tls encryption
- Restrict access between pods

Authentication

- Kubernetes does not handle users internally
    - Serviceaccounts can be managed with kubernetes
- Kube-apiserver auth mechanisms
    - Static password file
        - Eg. Csv. File with pw, username, userid, [group] and specify file in kubeapiserver service
        - Not recommended
    - Static token file
        - Instead pw use a token
        - Not recommended
    - Certificates
    - 3rd party identity services

TLS Basics

- Guarantee trust between parties in a transaction
- Asymmetric encryption(public key and private key) PKI(public key infrastructure)
    - Generate with ssh-keygen
    - On Server: openssl genrsa –out <key> <base>
    - Certificate to verify validity of key transfer (client and server certificates)
    - CA organizations sign certificates with their public and private keys (root certificates)
        - You can host private CA instances
    - Browsers check certificates
    - Server can request from client

TLS in Kubernetes

- Communication between nodes, components and users needs to be secured
- Servers:
    - Kube-apiserver exposes http service and requires server certificate
    - Etcd-server has requires own server certificate
    - Kubelet servers require server certificates too
- Clients:
    - Users like admin
    - Kube-scheduler
    - Kube-controller-manager
    - Kube-proxy
    - Kube-apiserver with etcd-server
    - Kube-apiserver with kubelet-server

TLS in Kubernetes – Certificate Creation

- Different tools like easyrsa, **openssl**, cfssl
- Kubeadm does most of the work for you
- CA Certificate:
    - Generate Key: *openssl genrsa –out <key-name> 2048*
    - Certificate Signing Request*: openssl req –new –key <keyname> -subj "/CN=KUBERNETES-CA" –out <keyfile>*
    - Sign Certificates: *openssl x509 –req –in <keyfile> -signkey <keyname> -out <certname>*
    - CA certificates files have to be well protected
- Client Certificates:
    - Admin user:
        - Like CA certificate with different names and "CN=kube-admin/O=system:masters"
        - Openssl x509 –req –in admin.csr –CA ca.crt –Cakey ca.key –out admin.crt
        - Specifiy certificate in kube-config.yaml
    - Similar process for other client certificates
- Server Certificates
    - Etcd-server:
        - Similar to other certificates
        - Needs additional peer certificates
    - Kube-apiserver:
        - Create openssl.cnf with dns and IP's and path as option while generating
    - Kubelet-server:
        - Named after nodes and use them in kubelet-config.yaml
        - E.g.: Name: system:node:node01
        - Needs to be done for each node

View Certificate Details

- See spreadsheat in directory for what to check
- Cat /etc/kubernetes/manifests/kube-apiserver.yaml
- *openssl x509 –in <cert file> -text –noout* to view details
- inspect service logs for problems if kubeapi or etcd is down use docker logs

Certificates API

- Certificate signing request is send trough certificates API to CA server
    - Admin can approve request and share cert with user
    - Controller Manager does this
    - *Openssl genra –out <key> 2048*
    - *Openssl req –new –key <key> -subj "/CN=<username>" –out <file>*
    - Request specified with yaml file
    - *Kubectl get csr <request> -o yaml*
    - *Kubectl certificate approve <request>*

Kube Config

- Use kubeConfig file to store cert and key configurations and use the file for api requests
- 3 Sections:
    - Clusters: clusters you have access too
    - Users: users accounts you have you
    - Contexts: specify which user you want to use for which cluster
        - Namespace can also specified here
- *Kubectl config view [-- <config file>]*
- *Kubectl config use-context <user@cluster>*

API Groups

- Core group (/api):
    - Namespaces, pods, events, nodes, bindings, …
- Named group (/apis):
    - /apps, /extensions, /networking.k8s.io, … with their resources with their actions
    - See them under *curl <server-adress>:6443-k*
        - Use certificate files to authenticate
        - Or use *kubectl proxy*

Role Based Access Control

- Specified in role definition yaml
    - Role binding yaml object needs to be specified too
- *Kubectl get/describe roles*
- *Kubectl get/ describe rolebindings*
- Check access with:
    - kubectl auth can-i <do something>
    - kubectl auth can-i <do something> as <user>

Cluster Roles and Role Bindings

- see namespaced resources: kubectl api-resources –namespaced=true
- cluster roles/bindings for cluster scoped resources
- similar to non-cluster based roles
- can be used for namespace based resources too

Image Security

- hosting private image registry/registry is good for security
- kubectl create secret docker-registry <name>
    - specify secret under imagePullSecrets

Security Contexts

- container security settings can be specified in pod level
    - add securityContext under spec in definition file or under container

Network Policy

- PODs can communicate by default with each other
    - Can be prevented with network policies linked to PODs
        - Create with definition file and link with labels

Storage (S8)

Networking (S9)

Install Hard Way (S11)

Troubleshooting (S12)

Other Topics (S13)

Mock Exams (S14)