1. Know how to configure authentication and authorization

Authentication and authorization

- Create two namespaces, one for production and the other for development.
- Create a new user DevDan and assign a password of lfs258.
- Generate a private key then Certificate Signing Request (CSR) for DevDan.
- Using the newly created request, generate a self-signed certificate using the x509 protocol. Use the CA keys for the Kubernetes cluster and set a 45 day expiration.
- Update the access config file to add user DevDan and reference the new key and certificate (using set-credentials)
- Create context name DevDan-context with cluster kubernetes, namespace development, and user DevDan
- Create a role name Developer in development namespace:
    - apiGroups: "", "extensions", "apps"
    - resources: "deployments" , "replicasets" , "pods"
    - verbs: ["*"]
- Create a rolebinding name developer-role-binding in development namespace: role developer and user DevDan
- In DevDan-context, create a new deployment nginx with image nginx, verify it exists, then delete it
- Create context name ProdDan-context with cluster kubernetes, namespace production, and user DevDan
- Create a role name dev-prod in production namespace:
    - apiGroups: "", "extensions", "apps"
    - resources: "deployments" , "replicasets" , "pods"
    - verbs: "get", "list", "watch"
- Create a rolebinding name production-role-binding in production namespace: role dev-prod and user DevDan

show

- Create two namespaces

```
kubectl create ns production
kubectl create ns development
```

- Create a new user and password

```
sudo useradd DevDan -s /bin/bash
sudo passwd DevDan
```

- Generate private key then csr for DevDan

```
openssl genrsa -out DevDan.key 2048
openssl req -new -key DevDan.key -out DevDan.csr -subj "/CN=DevDan/O=development"
# To view the csr
openssl req -in DevDan.csr -text -noout
```

- Generate the self-signed certificate

```
sudo openssl x509 -req -in DevDan.csr -out DevDan.crt \
       -CA /etc/kubernetes/pki/ca.crt \
       -CAkey /etc/kubernetes/pki/ca.key \
       -CAcreateserial -days 45
# To view the certificate
openssl x509 -in DevDan.crt -text -noout
```

- Update the access config file to add user DevDan

```
kubectl config set-credentials DevDan --client-certificate=/home/lkd/DevDan.crt --client-key=/home/lkd/DevDan.key
```

- Create context name DevDan-context

```
kubectl config set-context DevDan-context --user=DevDan --cluster=kubernetes --namespace=development
kubectl config get-contexts
```

- Create role 'developer'

```
kubectl create role --resource=pods --verb="*" --dry-run -o yaml > role-dev.yaml
vim role-dev.yaml
```

```
kubectl create -f role-dev.yaml

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
    name: developer
    namespace: development
rules:
- apiGroups: ["", "extensions", "apps"]
    resources: ["pods", "deployments", "replicasets"]
    verbs: ["*"]
```

- Create rolebinding 'developer-role-binding'

```
kubectl create rolebinding developer-role-binding --role=developer \
          --user=DevDan -n development --dry-run -o yaml > developer-role-binding.yaml
kubectl create -f developer-role-binding.yaml
```

- In DevDan-context, create a new deployment nginx with image nginx, verify it exists, then delete it

```
kubectl --context=DevDan-context create deployment nginx --image=nginx
kubectl --context=DevDan-context get deployment
kubectl --context=DevDan-context delete deployment nginx
```

2. Understand Kubernetes security primitives

Certificate API

- Create a new user DevTrang with password lfs258 (list all users with cat /etc/passwd)
- Create DevTrang.key and DevTrang.csr with openssl.
- Create a Certificate signing request (csr) object from DevTrang.csr to send to the Kubernetes API (here).
- Get csr and approve it.
- Download the certificate and use it.

show

- Create new user and password

```
sudo useradd DevTrang -s /bin/bash
sudo passwd
```

- Create private key and csr

```
openssl genrsa -out DevTrang.key 2048
openssl req -new -key DevTrang.key -out DevTrang.csr --subj "/CN=DevTrang/O=development"
```

- Create csr object in Kubernetes
    - Search kubernetes.io for csr here.
    - Copy and paste to terminal, edit to use DevTrang.csr

```
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
    name: DevTrang
spec:
    request: $(cat DevTrang.csr | base64 | tr -d '\n')
    usages:
    - digital signature
    - key encipherment
    - server auth
EOF
```

- Get csr and approve it.

```
kubectl get csr
kubectl certificate approve DevTrang
```

- Download and use it.

```
kubectl get csr DevTrang -o jsonpath='{.status.certificate}' | base64 --decode > DevTrang.crt
```

Kubeconfig

show

Cluster roles and cluster role bindings

- A new user DevDan joined the team. He will be focusing on the nodes in the cluster:
  - Create the required ClusterRoles and ClusterRoleBindings
  - So he gets access to the nodes (Grant permission to list nodes)
- Create the required ClusterRoles and ClusterRoleBindings to allow DevDan access to Storage.
  - ClusterRole: storage-admin, Resource: persistentvolumes, Resource: storageclasses
  - ClusterRoleBinding: DevDan-storage-admin, ClusterRoleBinding Subject: DevDan, ClusterRoleBinding Role: storage-admin

show

- Create clusterrole and clusterrolebinding to list nodes:

```
kubectl create clusterrole DevDan-cluster-role --verb=list --resource=nodes --dry-run -o yaml > DevDan-cluster-role.yaml
kubectl create -f DevDan-cluster-role.yaml
kubectl create clusterrolebinding DevDan-role-binding --user=DevDan --clusterrole=DevDan-cluster-role
# Check the authorization
kubectl auth can-i --list --as DevDan
kubectl auth can-i list nodes --as DevDan
```

- Create the required ClusterRoles and ClusterRoleBindings to allow DevDan access to Storage.

```
kubectl create clusterrole storage-admin --resource=persistentvolumes --resource=storageclasses --verb="*"
kubectl create clusterrolebinding DevDan-storage-admin --user=DevDan --clusterrole=storage-admin
# Check the authorization
kubectl auth can-i --list --as DevDan
```

3. Know how to configure network policies

Reference is here. Study the kubernetes network policy recipes here.

Network Policy

- Create a network policy to allow traffic from the 'Internal' application only to the 'payroll-service' and 'db-service'
- Policy Name: internal-policy, Policy Types: Egress
- Egress Allow: payroll, Payroll Port: 8080
- Egress Allow: mysql, MYSQL Port: 3306

show

4. Create and manage TLS certificates for cluster components

View the certificates

- Identify the certificate file used for the kube-api server.
- Identify the Certificate file used to authenticate kube-apiserver as a client to ETCD Server.
- Identify the key used to authenticate kubeapi-server to the kubelet server.
- Identify the ETCD Server Certificate used to host ETCD server.

- Identify the ETCD Server CA Root Certificate used to serve ETCD Server (ETCD can have its own CA. So this may be a different CA certificate than the one used by kube-api server).
- What is the Common Name (CN) configured on the Kube API Server Certificate? (OpenSSL Syntax: openssl x509 -in file-path.crt -text -noout).
- Which of the below alternate names is not configured on the Kube API Server Certificate?
- How long, from the issued date, is the Root CA Certificate valid for? (File: /etc/kubernetes/pki/ca.crt)

show

5. Work with images securely

Reference is here.

Image security

- Create a deployment name web with image nginx:alpine, scale to replicas=2
- Use a modified version of the application from an internal private registry. Update the image of the deployment to use a new image from myprivateregistry.com:5000
- Create a secret object with the credentials required to access the registry:
    - Name: private-reg-cred
    - Username: dock_user
    - Password: dock_password
    - Server: myprivateregistry.com:5000
    - Email: dock_user@myprivateregistry.com
- Configure the deployment to use credentials from the new secret to pull images from the private registry

show

- Create and scale deployment

```
kubectl create deployment web --image=nginx:alpine
kubectl scale deployment web --replicas=2
kubectl edit deployment web
```

- Update deployment using new private registry

```
    template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - image: myprivateregistry.com:5000/nginx:alpine
```

- Create a secret object:

```
kubectl create secret docker-registry private-reg-cred --docker-server=myprivateregistry.com:5000 \
          --docker-username=dock_user --docker-password=dock_password --docker-email=dock_user@myprivateregistry.comuser@myprivateregistry.com
```

- Configure the deployment to use credentials from the new secret:

```
kubectl edit deployment web

    template:
    metadata:
      labels:
        app: web
    spec:
      imagePullSecrets:
      - name: private-reg-cred
      containers:
      - image: myprivateregistry.com:5000/nginx:alpine
```

6. Define security contexts

Reference is here.

Note:

- **You may choose to configure the security settings at a container level or at a pod level.**
- **The User ID defined in the securityContext of the container overrides the User ID in the POD.**

Security Context

- Create a pod 'ubuntu-sleeper' with image 'ubuntu', command 'sleep 4800'
- What is the user used to execute the sleep process within the 'ubuntu-sleeper' pod?
- Edit the pod 'ubuntu-sleeper' to run the sleep process with user ID 1010.
- Edit the pod 'ubuntu-sleeper'
    - Add another container name 'sidecar', image 'ubuntu', command sleep 5000
    - Add securityContext to pod level, runAsUser 1001
    - Check who run container 'ubuntu-sleeper', who run 'sidecar'
- Update pod 'ubuntu-sleeper', remove sidecar, to run as Root user and with the 'SYS_TIME' capability.
    - Now try to run the below command in the pod to set the date (date -s '19 APR 2012 11:14:00')

show

- Create a pod 'ubuntu-sleeper'

```
kubectl run ubuntu-sleeper --generator=run-pod/v1 --image=ubuntu --dry-run -o yaml > ubuntu-sleeper.yaml
vim ubuntu-sleeper.yaml
kubectl create -f ubuntu-sleeper.yaml
```

```
spec:
    containers:
    - image: ubuntu
      name: ubuntu-sleeper
      command:
      - sleep
      - "4800"
```

- What is the user used to execute the sleep process within the 'ubuntu-sleeper' pod?

```
kubectl exec ubuntu-sleeper -it --sh
ps aux
```

- Edit the pod 'ubuntu-sleeper' to run the sleep process with user ID 1010.

```
spec:
    containers:
    - image: ubuntu
      name: ubuntu-sleeper
      command:
      - sleep
      - "4800"
      securityContext:
        runAsUser: 1010
```

- Edit the pod 'ubuntu-sleeper' to multi-container pod

```
spec:
    securityContext:
      runAsUser: 1001
    containers:
    - image: ubuntu
      name: ubuntu-sleeper
      command:
      - sleep
      - "4800"
      securityContext:
      resources: {}
```

```
      runAsUser: 1010
  - image: ubuntu
    name: sidecar
    command: ["sleep","5000"]
```

Check who run the container

```
kubectl exec ubuntu-sleeper -it -c sidecar -- sh
ps aux
kubectl exec ubuntu-sleeper -it -c ubuntu-sleeper -- sh
ps aux
```

- Add capability

```
spec:
    containers:
    - image: ubuntu
      name: ubuntu-sleeper
      command:
      - sleep
      - "4800"
      securityContext:
        capabilities:
          add: ["SYS_TIME"]

kubectl exec -it ubuntu-sleeper -- date -s '19 APR 2012 11:14:00'
```

7. Secure persistent key value store

show