Security:

→> the first line of defense is to protect the Kube API server
-this defense can be designed around 2 critical questions: who can access the cluster? And what can they do?

- but what about the accesses within the Cluster?
- All the pods is a cluster can access all other pods in the cluster
- The access between them can be restricted using network policies.

- considering the account that need or can access the cluster, there can be user accounts (for example: admin, developers etc) or service accounts.
- Kubernetts doesnot have support for access to users accounts natively, this can however be achieved by using external source, like file with user details, certificates, or third party id services like LDAP to manage users.
- Kubernetes can however manage service accounts.

- All requests in a cluster are handled by API server
- The API server authenticates the request and then processes the request.

How does API server authenticate a user request?

- an API server can have a static file with username and password or it can have a static file with user name and token.
- The authentication can also be done using certificates, or services like LDAP or Kerberos.

- for the case using static password or token file, the file is passed onto the apiserver during its start as one of the command line argument. The argument is --basic-auth-file
- This above argument can be supplied in the manifest file or the service definition of the kube API server

- For the user to be authenticated and to make a successful request., require like following can be made:

```
curl -v -k "https://<ip>:6443/api/v1/pods
                              -u "user1:password1" ↵
```

- for the case with token, we can provide the token as bearer token in the header of the request made to the API server.

- These methods of using password or token in a static file is not a recommended approach as they can be insecure. These techniques are is anyway depricated in kube v1.19

TLS basics

- symmetric encryptions: same key is used both for encryption and decryption.
- since the same key is used there is a chance hacker getting access to the whole they
- To mitigate this issue a assymentric encryption method is used to share the private key
- Asymmetric encryptions: a pair of keys is used, one for encryption and other for decryptions.
- the key pair consists of private and public key
- the public key can be accessed by anyone
- anything that is encrypted by public key can only be accessed by decrypting with the private key

Even for the case with asymmetric key encryption where a user's data is encrypted with public key of a server, a hacker could intercept the communication and provide their public key impersonating to be the server. This is where comes the certificate authority.

When the server sends their public key, they will not send the key alone; but they Will Send a certificate that has the key in it. The certificate has information like:

- who the issuer is
- Who the cert is issued to
- Validity dates

But anyone can create a certificate like that with all those info. This where the signing of certificate comes into play.
  - if you generate a certificate and sign it, it is called self signed certificate; which is not a very unsafe way to do things
  - you can also get certificate signed by some external authority called CA.
    - this can be done by creating CSR or certificate signing request using the public key, and the domain name of your website.
    - this can be done using OpenSSL
    - the CA validates the info , signs and sends back the cert.

The certificates are signed by the CA's private key and their public keys are built into major browsers. The browser uses the public key to validate the certificates signed by CA. This way no hacker can impersonate being a CA.

The above discussions so far is for public websites; for private websites like company internal sites, the company can host private CA. This public key of the private CA server can be deployed in all the employees browser.

TLS in kubernetes:

- the server uses asymmetric encryption using a public and private key pair.
  - This keys are called "server keys",
  - the certificate of the server contains the public key.
  - This certificate is called "server certificate".
- the public key of the CA is embedded into the certificate of the CA.
- - this is called "root certificate".
- The client will also have its certificate and a private key, called "client certificates"

- certificates with public key are with extension .pem .crt .
- Private key are with extension .key or with -key.pem in their name.

Every communication in Kube needs to be verified. A server uses "server certificates" and all clients use " client certificate" to verify who they say they are.

Kube API server:
- exposes an HTTP endpoint .
- Has apiserver.crt and apiserver.key.

Similarly, ETCD server also has etcdserver.Crt and .key files.

Each worker nodes have kubelet server running on them. They also expose http endpoint and have the certificate and by files.

We as a user need to make request to the API server using clients like kubectl. The user certificate and key are, was name them admin.key and admin.crt.

Similarly kube scheduler is also a client to the APA server which has its own certificate and key.

Similar is the case for Kube controller manager.

Similarly kube proxy also requires certificate and key to validate its request to the API server.

API server also talks to ETCD server and kubelet.it can use its server certificate to authenticate communication or generate new certificate and private key to talk to ETCD server and kubelet.

Steps to create certificates:
- generate keys using OpenSSL, a private key
- Create CSR using the private key above
- Sign the certificate using X509 OpenSSL command. This creates self signing certificates.
- We will use this certificate as CA certificate.

To create certificates for the client, we follow the same process of creating private key and CSR but supply CA certificate and key to sign them. The clients can use this certificate to authenticate itself when making call to API server.

More on notes