Authentication

You can enable multiple authentication methods at once. You should usually use at least two methods:

- **service account tokens** for service accounts - Enabled by default
- at least one other method for **user authentication**.
    - Client certs
    - OpenID connect tokens
    - Bootstrap Tokens - Beta
    - Static token file
    - Static password file

users/groups are not represented as objects in kubernetes

Serviceaccount
https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/

- represented as object
- Used by pods/applications to access API servers
- Have associated **secrets and generated automatically and mounted automatically to POD**
  To disable → automountServiceAccountToken: false
  In service accoount yaml or POD yaml..

    - **The service account has to exist at the time the pod is created, or it will be rejected.**
    - **You cannot update the service account of an already created pod.**
    - **Any tokens for non-existent service accounts will be cleaned up by the token controller.**
- Secrets contains
    - Jwt TOKEN -- **Created and signed by using the service account key pair**
        - **service-account-key.pem**
        - **Service-account.pem**
        - Controller manager generated tokens..
    - Service account cert (service-account.pem)
        - Required for validating the JWT token…

A Service account admission controller

1. - adding default service account to every pod if not present
2. - mounting secret as volume

Token controller -

1. creates new token/secrets for new service account..Also takes care of deletion..

Service account controller-

1. ensures every new namespace have a default service account

Service Account tokens

- Enabled by default
- Service account needs to pass signed tokens to authenticate
- Request authenticated using service-acccount keys
- While starting API server below param needs to be passed :

  --service-account-key-file A file containing a PEM encoded key for signing bearer tokens. **If unspecified, the API server's TLS private key will be used.**

  Service accounts authenticate with the username system:serviceaccount:(NAMESPACE):(SERVICEACCOUNT), and are assigned to the groups system:serviceaccounts and system:serviceaccounts:(NAMESPACE).

Client certs

- enabled by passing the **--client-ca-file=SOMEFILE** option to API server
- SOMEfile - CA certs that will validate the client certs. There can be one or more CA certs file specified when starting API server.
- Hence client need to specify which authority granted the cert…
- Client cert -- > CN field == UserName
  O field   == Group name, user can be part of more than one group..

  ```
  openssl req -new -key jbeda.pem -out jbeda-csr.pem -subj "/CN=jbeda/O=app1/O=app2"
  ```

OpenID connect tokens

Bootstrap Tokens - Beta

Static token file

- Enabled by passing **--token-auth-file=SOMEFILE** option to API server.
- SOMEFILE - static CSV file containing below contents..
  - **token,user,uid,"group1,group2,group3"**

➔ Token file edit - needs API server restart

➔ Tokens do not expire, last indefinitely..

➔ Add the token in HTTP header

    - **Authorization: Bearer 31ada4fd-adec-460c-809a-9e56ceb75269**

Static password file

- Enabled by passing **--basic-auth-file=SOMEFILE** option to API server.
- SOMEFILE - static CSV file containing below contents..
    - **token,user,uid,"group1,group2,group3"**
- password edit - needs API server restart
- password do not expire, last indefinitely..
- Add the token in HTTP header
    - **Authorization: Basic BASE64ENCODED(USER:PASSWORD)**

Security Context

1. A security context defines privilege and access control settings for a Pod or Container
2. Security settings that you specify for a Container apply only to the individual Container, and they override settings made at the Pod level when there is overlap.

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

runAsUser=405

- The UID to run the entrypoint of the container process. Defaults to user specified in image metadata if unspecified. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.

runAsGroup

- The GID to run the entrypoint of the container process. Uses runtime default if unset. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.
- The runAsGroup field specifies the primary group ID of 3000 for all processes within any containers of the Pod. If this field is omitted, the primary group ID of the containers will be root(0).

runAsNonRoot=true

- Indicates that the container must run as a non-root user.
- If true, the Kubelet will validate the image at runtime to ensure that it does not run as UID 0 (root) and fail to start the container if it does. If unset or false, no such validation will be performed. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.
- Error:
  - Error: container has runAsNonRoot and image will run as root

privileged=tue

- Run container in privileged mode. Processes in privileged containers are essentially equivalent to root on the host. Defaults to false.
- Pods default ---- > has root privileges only on the container process

```
imsrv01@imsrv01-Lenovo-ideapad-700-15ISK:~/CKA/cka/security$ kubectl exec -it
pod-with-defaults -- ls /dev
```

```
core               null               shm                termination-log
fd                 ptmx               stderr             tty
full               pts                stdin              urandom
mqueue             random             stdout             zero
imsrv01@imsrv01-Lenovo-ideapad-700-15ISK:~/CKA/cka/security$ k exec -it
privileged-pod -- ls /dev
autofs             stdin              tty48
bsg                stdout             tty49
btrfs-control      termination-log    tty5
core               tty                tty50
cpu                tty0               tty51
cpu_dma_latency    tty1               tty52
fd                 tty10              tty53
full               tty11              tty54
fuse               tty12              tty55
hpet               tty13              tty56
hwrng              tty14              tty57
input              tty15              tty58
kmsg               tty16              tty59
loop-control       tty17              tty6
loop0              tty18              tty60
loop1              tty19              tty61
loop2              tty2               tty62
loop3              tty20              tty63
loop4              tty21              tty7
loop5              tty22              tty8
loop6              tty23              tty9
loop7              tty24              ttyS0
mapper             tty25              ttyS1
mem                tty26              ttyS2
memory_bandwidth   tty27              ttyS3
mqueue             tty28              urandom
net                tty29              usbmon0
network_latency    tty3               vboxguest
network_throughput  tty30             vboxuser
null               tty31              vcs
nvram              tty32              vcs1
port               tty33              vcs2
ptmx               tty34              vcs3
pts                tty35              vcs4
random             tty36              vcs5
rfkill             tty37              vcs6
rtc0               tty38              vcsa
sda                tty39              vcsa1
sda1               tty4               vcsa2
sda2               tty40              vcsa3
sg0                tty41              vcsa4
sg1                tty42              vcsa5
```

```
shm                  tty43                vcsa6
snapshot             tty44                vga_arbiter
snd                  tty45                vhost-vsock
sr0                  tty46                zero
stderr               tty47
```

Linux Capabilities

- The capabilities to add/drop when running containers. Defaults to the default set of capabilities granted by the container runtime.
- With Linux capabilities, you can grant certain privileges to a process without granting all the privileges of the root user.
- Find capabilities
  ```
  cd /proc/1  (1 - proces id)
  cat status
  ```

```
    securityContext:
      capabilities:
        add:
        - SYS_TIME
```

```
    securityContext:
      capabilities:
        drop:
        - CHOWN
```

ReadonlyRootFileSystem

- Whether this container has a read-only root filesystem. Default is false

```
securityContext:
      readOnlyRootFilesystem: true
    volumeMounts:
    - name: my-volume
      mountPath: /volume
      readOnly: false
```

Cannot write to root file system but can write to mounted volume..

SELinux

```
...
securityContext:
  seLinuxOptions:
    level: "s0:c123,c456"
```

AppArmour

AppArmor is a Linux kernel security module that supplements the standard Linux user and group based permissions to confine programs to a limited set of resources. AppArmor can be configured for any application to reduce its potential attack surface and provide greater in-depth defense. It is configured through profiles tuned to whitelist the access needed by a specific program or container, such as Linux capabilities, network access, file permissions, etc. Each profile can be run in either *enforcing* mode, which blocks access to disallowed resources, or *complain* mode, which only reports violations.

Secure Images

The default pull policy is IfNotPresent which causes the Kubelet to skip pulling an image if it already exists. If you would like to always force a pull, you can do one of the following:

- set the imagePullPolicy of the container to Always.
- omit the imagePullPolicy and use :latest as the tag for the image to use.
- omit the imagePullPolicy and the tag for the image to use.
- enable the AlwaysPullImages admission controller.

Note that you should avoid using :latest tag

Access private repositories

Example : Accessing docker/ecr repo using service account with secrets - imagePullSecrets

Authorization

Modes

Node - A special-purpose authorization mode that grants permissions to kubelets based on the pods they are scheduled to run

The Node authorizer allows a kubelet to perform API operations. This includes:

Read operations:

- services
- endpoints
- nodes
- pods
- secrets, configmaps, persistent volume claims and persistent volumes related to pods bound to the kubelet’s node

Write operations:

- nodes and node status (enable the NodeRestriction admission plugin to limit a kubelet to modify its own node)
- pods and pod status (enable the NodeRestriction admission plugin to limit a kubelet to modify pods bound to itself)
- events

Auth-related operations:

- read/write access to the certificationsigningrequests API for TLS bootstrapping
- the ability to create tokenreviews and subjectaccessreviews for delegated authentication/authorization checks

RBAC -

- Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise
- To enable RBAC, start the apiserver with --authorization-mode=RBAC

Webhook -

- A WebHook is an HTTP callback: an HTTP POST that occurs when something happens; a simple event-notification via HTTP POST. A web application implementing WebHooks will POST a message to a URL when certain things happen

ABAC - Attribute-based access control (ABAC) defines an access control paradigm whereby **access rights are granted to users through the use of policies which combine attributes together**. The policies can use any type of attributes (user attributes, resource attributes, object, environment attributes, etc).

1. Alice can do anything to all resources:
   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "alice", "namespace": "*", "resource": "*", "apiGroup": "*"}}
   ```
2. The Kubelet can read any pods:
   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet", "namespace": "*", "resource": "pods", "readonly": true}}
   ```
3. The Kubelet can read and write events:
   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet", "namespace": "*", "resource": "events"}}
   ```
4. Bob can just read pods in namespace “projectCaribou”:
   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "bob", "namespace": "projectCaribou", "resource": "pods", "readonly": true}}
   ```

Anyone can make read-only requests to all non-resource paths:

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"group": "system:authenticated", "readonly": true, "nonResourcePath": "*"}}
```

5.
   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"group": "system:unauthenticated", "readonly": true, "nonResourcePath":
   ```

For example, if you wanted to grant the default service account (in the kube-system namespace) full privilege to the API using ABAC, you would add this line to your policy file:

```
{"apiVersion":"abac.authorization.kubernetes.io/v1beta1","kind":"Policy","spec":{"user":"system:serviceaccount:kube-system:default","namespace":"*","resource":"*","apiGroup":"*"}}
```

- --authorization-mode=ABAC Attribute-Based Access Control (ABAC) mode allows you to configure policies using local files.
- --authorization-mode=RBAC Role-based access control (RBAC) mode allows you to create and store policies using the Kubernetes API.
- --authorization-mode=Webhook WebHook is an HTTP callback mode that allows you to manage authorization using a remote REST endpoint.
- --authorization-mode=Node Node authorization is a special-purpose authorization mode that specifically authorizes API requests made by kubelets.
- --authorization-mode=AlwaysDeny This flag blocks all requests. Use this flag only for testing.
- --authorization-mode=AlwaysAllow This flag allows all requests. Use this flag only if you do not require authorization for your API requests.

SelfSubjectAccessReview

```
kubectl auth can-i
```

- for quickly querying the API authorization layer.
- uses the SelfSubjectAccessReview API to determine if the current user can perform a given action, and works regardless of the authorization mode used

```
kubectl auth can-i create deployments --namespace dev
yes
```

```
kubectl auth can-i create deployments --namespace prod
no
```

Others:

- **SubjectAccessReview -** Access review for any user, not just the current one. Useful for delegating authorization decisions to the API server. For example, the kubelet and extension API servers use this to determine user access to their own APIs.
- **LocalSubjectAccessReview -** Like SubjectAccessReview but restricted to a specific namespace.
- **SelfSubjectRulesReview -** A review which returns the set of actions a user can perform within a namespace. Useful for users to quickly summarize their own access, or for UIs to hide/show actions.

Userimpersonation

```
$kubectl auth  can-i create deployment -n kube-system --as default
```

no - no RBAC policy matched

Network Policies

Creating TLS certificates

Admission Controller

`AlwaysPullImages`

- This admission controller modifies every new Pod to force the image pull policy to Always.
- This is useful in a multitenant cluster so that users can be assured that their private images can only be used by those who have the credentials to pull them.
- When this admission controller is enabled, images are always pulled prior to starting containers, which means valid credentials are required