K8S (4) AuthN/AuthZ/AdmissionControl to manage request to API Server

Authentication (AuthN):

Make sure the user is who he claim he is.
K8S doesn't have user object. Using username to manage API access.
Two types of username (1) Normal User (2) Service Account
AuthN modules:
(1) X509 Client Certificates: **--client-ca-file=[certificated file]**
(2) Static Token File: **--token-auth-file=[Bear token file]**
(3) Bootstrap Tokens: token used for bootstrapping new K8S cluster
(4) Service Account Tokens: Automatically enabled authenticators that use signed bearer tokens to verify requests. These tokens get attached to Pods using the Service Account Admission Controller, which allows in-cluster processes to talk to the API server.
(5) OpenID Connect Tokens: external OAuth2 provider will verify (Google/Salesforce)
(6) Webhook Token Authentication: remote service will verify the bear token
(7) Authenticating Proxy: Programming extra authN logic

Authorization (AuthZ):

After authN, what can this user do?
K8S has couple of modules to authZ API request. We can config multiple modules at the same time.
(1) Node: API authorized by kubelet
(2) Attribute-Based Access Control (RBAC): **--authorization-mode=ABAC --authorization-policy-file=PolicyFile.json**

```
# PolicyFile.json
# user bob can only read Pods in the Namespace lfs158
{
  "apiVersion": "abac.authorization.kubernetes.io/v1beta1",
  "kind": "Policy",
  "spec": {
    "user": "bob",
    "namespace": "lfs158",
    "resource": "pods",
    "readonly": true
  }
}
```

(3) Webhook: authZ by remote service **--authorization-webhook-config-file=[configuration of the remote authorization service]**
(4) Role-Based Access Control (RBAC): **--authorization-mode=RBAC**
Resource accessing operation (create/get/update/patch) is based on Role.
Multiple roles can associated with a Normal User or Service Account.
Two types of roles: (1) Role: grant access to specific namespace (2) ClusterRole: scope is cluster-wide

```
# role.yaml
# Defines a pod-reader role, which has access only to read the Pods of lfs158
Namespace
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: lfs158
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Once we have role in place, we need RoleBinding to associate role with a user.

Two types of roleBinding: (1) RoleBinding: Associate user to a role, permission only in role's namespace (2) ClusterRoleBinding: Grant access at cluster-level, to ALL namespace

```
# roleBinding.yaml
# Defines a bind between the pod-reader Role and user bob, user bob can only
read the Pods in lfs158 Namespace.
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-read-access
  namespace: lfs158
subjects:
- kind: User
  name: bob
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Admission Control:

A module to validate user's request.
There are two types of admission controls: (1) Validating (Checking) and (2) mutating (Changing).
There are Granular Access Control Policies (who has access to which resource). These policies is managed by admission controllers.
Flag is **--enable-admission-plugins=NamespaceLifecycle,ResourceQuota,PodSecurity,DefaultStorageClass**

Demo of authN and authZ:

Fist we need to create a new user. The scenario is: your have new coming devOps (My. CY) need permission to access the cluster.

```
$ kubectl create namespace ns-for-demo
namespace/ns-for-demo created

$ mkdir rbac
$ cd rbac
$ openssl genrsa -out CY.key 2048 // use openssl to creat a private key for user CY
$ openssl req -new -key CY.key -out CY.csr -subj "/CN=CY/O=learner" // generate
certificate signing request(CSR, based on your private key) for user CY
$ ls
CY.csr CY.key

$ cat CY.csr | base64 // Encoding csr in base64
LS0tLS1CRUdJ...VC0tLS0tCg==

$ cat signing-request.yaml
// this will create a CertificateSigningRequest object in K8S
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
name: CY-csr
spec:
groups:
- system:authenticated
request: LS0tLS1CRUdJ...LS0tCg== // what you got from encoding CY.csr file
signerName: kubernetes.io/kube-apiserver-client
usages:
- digital signature
- key encipherment
- client auth

$ kubectl create -f signing-request.yaml
certificatesigningrequest.certificates.k8s.io/CY-csr created

$ kubectl get csr // The csr object we just created is here, but in pending state
NAME AGE SIGNERNAME REQUESTOR REQUESTEDDURATION CONDITION
CY-csr 8s kubernetes.io/kube-apiserver-client minikube-user <none> Pending

$ kubectl certificate approve CY-csr // approved this csr
certificatesigningrequest.certificates.k8s.io/CY-csr approved

$ kubectl get csr
NAME AGE SIGNERNAME REQUESTOR REQUESTEDDURATION CONDITION
CY-csr 29s kubernetes.io/kube-apiserver-client minikube-user <none> Approved,Issued

$kubectl get csr CY-csr -o jsonpath='{.status.certificate}' | base64 -d > CY.crt
// Extract the Approved certificate from csr object, decode with base64 and save it
as certificate file (CY.crt)
```

```
$ cat CY.crt
-----BEGIN CERTIFICATE-----
MIIDFDCCAfygAwIBAgIQULkzZ5JCzp/2/SQa122OozANBgkqhkiG9w0BAQsFADAV
MRMwEQYDVQQDEwptaW5pa3ViZUNBMB4XDTIzMTIyMzE4NDYyOVoXDTI0MTIyMjE4
NDYyOVowHzEQMA4GA1UEChMHbGVhcm5lcjELMAkGA1UEAxMCQ1kwggEiMA0GCSqG
SIb3DQEBAQUAA4IBDwAwggEKAoIBAQDj2q/Kj0OZhHr2YtjKy58xvkKrpculKsLs
RAJgsRllwwcms+0vbP58mmcbu2yqnw2LDp33arUddv18Ab7GqJ+mtpAS4urgXm43
HdR3Px5tH/kABwhlyBLj1MRWUzNupOYVWt7s2D+jzNqCxDgrZzY2NpPZ0jsmMWEu
By6knGzrttP79TiszZs2wwOUNeTgb7iOH2fFdXa5vrTeS/Kwe2ixuzGD17s+oPmM
6XVcPGIKzyac3vjo4Rhp6n+t8JsS2aRD4B1Qoaz9IUKt3H54Dd7jj4FA/2NmjUq3
ZYl0XV792EEc+WtqI9ICQJRKbVR00BVrK+8PcyQ983pP7CTQqmrrAgMBAAGjVjBU
MA4GA1UdDwEB/wQEAwIFoDATBgNVHSUEDDAKBggrBgEFBQcDAjAMBgNVHRMBAf8E
AjAAMB8GA1UdIwQYMBaAFDIqEWCiF4t7xaqWVaQ5nLmf0pUJMA0GCSqGSIb3DQEB
CwUAA4IBAQBpv+qDqoWhk82SsGyA/MmM0/m1065tYQDe6e+DZneB8yVgrCQT+8i+
j986M7xVUIMhUgk4mBjSwJ32md17lZ3m25ykl5ry9WldBzmd2MxDvM3/c28OjtlF
U5W4x6V1aOC7A7oQSZ1LuigSlff2+orrXv35mZ55qB6oX+4PNMarDttBZQLseJ7Q
oT+9K1fZSZ9EMr6QWO6jZ0Y3+YI5oYzZtZtSNoEXUlPJ8UAMpoElnfINzPMZ6hoQ
12GbitijIcGj7+o4PnlKjyJiK4aUT+ii6W9NR8NuVff9PK1P1AygX+mTS9Vm7FPL
Petpws58gOsj+eybyoviTn7CHP1ehBkc
-----END CERTIFICATE-----

$ kubectl config set-credentials CY --client-certificate=CY.crt --client-key=CY.key
// with the private key and approved certificate, we can create a new user
User "CY" set.

$ kubectl config set-context CY-context --cluster=minikube --namespace=ns-for-demo -
-user=CY
// create new context, associated with new user
Context "CY-context" created.

$ kubectl config view
// Besides the default context minikube and user minikube, our new CY context and CY
user has pssed the authN and added to minikube config
apiVersion: v1
clusters:
- cluster:
certificate-authority: /Users/chenyang/.minikube/ca.crt
extensions:
- extension:
last-update: Sat, 23 Dec 2023 01:10:39 PST
provider: minikube.sigs.k8s.io
version: v1.32.0
name: cluster_info
server: https://127.0.0.1:59108
name: minikube
contexts:
```

```
- context:
cluster: minikube
namespace: ns-for-demo
user: CY
name: CY-context
- context:
cluster: minikube
extensions:
- extension:
last-update: Sat, 23 Dec 2023 01:10:39 PST
provider: minikube.sigs.k8s.io
version: v1.32.0
name: context_info
namespace: default
user: minikube
name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: CY
user:
client-certificate: /Users/chenyang/Documents/myKubernetes/rbac/CY.crt
client-key: /Users/chenyang/Documents/myKubernetes/rbac/CY.key
- name: minikube
user:
client-certificate: /Users/chenyang/.minikube/profiles/minikube/client.crt
client-key: /Users/chenyang/.minikube/profiles/minikube/client.key
```

Next, we create a new pod. The new user CY doesn't not have permission to list the pod. But we can create a role and roleBinding to user CY.

```
$ kubectl create deployment nginx --image=nginx:alpine -n ns-for-demo
deployment.apps/nginx created
$ kubectl --context=CY-context get pods // CY-context has no permission
Error from server (Forbidden): pods is forbidden: User "CY" cannot list resource
"pods" in API group "" in the namespace "ns-for-demo"

// Create a pod-reader, with get/watch/list pod permission in ns-for-demo
$ cat role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
name: pod-reader
namespace: ns-for-demo
rules:
- apiGroups: [""]
```

```
resources: ["pods"]
verbs: ["get", "watch", "list"]
$ kubectl create -f role.yaml
role.rbac.authorization.k8s.io/pod-reader created
$ kubectl get roles -n ns-for-demo
NAME CREATED AT
pod-reader 2023-12-23T18:56:27Z

// assigns the permissions of the pod-reader Role to user CY
$ cat rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
name: pod-read-access
namespace: ns-for-demo
subjects:
- kind: User
name: CY
apiGroup: rbac.authorization.k8s.io
roleRef:
kind: Role
name: pod-reader
apiGroup: rbac.authorization.k8s.io
$ kubectl create -f rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/pod-read-access created
$ kubectl get rolebindings -n ns-for-demo
NAME ROLE AGE
pod-read-access Role/pod-reader 17s

// Everything in place, try to read the pod by CY-context
chenyang@ChenYangs-MBP rbac % kubectl --context=CY-context get pods
NAME READY STATUS RESTARTS AGE
nginx-b4ccb96c6-55nrg 1/1 Running 0 3m43s
```

Demo of Admission Control:

We choose **imagePullPolicy** to demo(**https://trstringer.com/kubernetes-alwayspullimages/**).
When the imagePullPolicy is "**IfNotPresent**", the kubelet will try to pull the image from cache on the node. If the image is not cached, then pull the image from the container registry.
When the imagePullPolicy is "**Always**", the kubelet always pulls the image from the container registry.

When the AlwaysPullImages admission control plugin is enabled in a cluster, this forces the image pull policy to be set to **Always.**
"**IfNotPresent**" policy could be a security breach, consider the following scenario

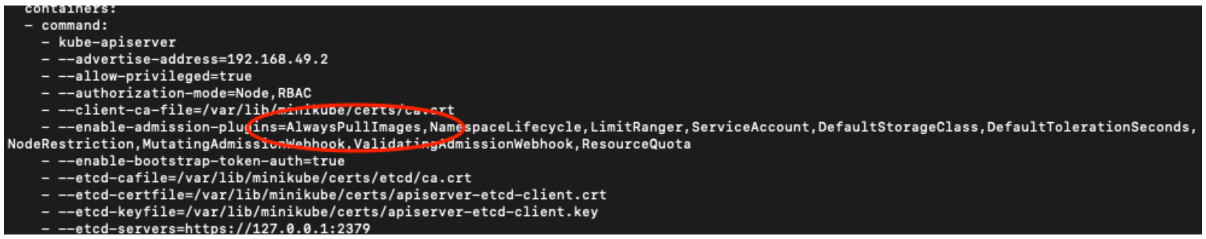
1. Admin1 creates **Pod1** that uses **SuperSecretImage1** by specifying **ImagePullSecret1** to gain access to the **SecureContainerRegistry** holding the image.
2. Kubernetes pulls down and caches **SuperSecretImage1** on the node and then creates **Pod1** accordingly.
3. Admin2, who does not have access to **ImagePullSecret1** or **SecureContainerRegistry**, attempts to create **Pod2** using **SuperSecretImage1** and an **imagePullPolicy** of **IfNotPresent**. Because this image is cached on the node, this operation is successful.
!! Admin2 got the secret he shouldn't access

```
$ kubectl describe pod kube-apiserver-minikube -n kube-system | grep -i admission //
Check what admission controller is enabled
--enable-admission-
plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTol
erationSeconds,NodeRestriction,MutatingAdmissionWebhook,ValidatingAdmissionWebhook,R
esourceQuota

$ kubectl run admitted --image=nginx --image-pull-policy=IfNotPresent // Create a
new pod with imagePullPolicy as IfNotPresent
pod/admitted created
$ kubectl get pod admitted -o yaml | grep -i imagepull // Check the current policy
imagePullPolicy: IfNotPresent

// Next we enable the admission controller, need to ssh into the minikube node
$ minikube ssh
docker@minikube:~$ sudo grep admission /etc/kubernetes/manifests/kube-apiserver.yaml
// Check the manifest definition for kube-apiserver pod
- --enable-admission-
plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTol
erationSeconds,NodeRestriction,MutatingAdmissionWebhook,ValidatingAdmissionWebhook,R
esourceQuota
docker@minikube:~$ sudo cp /etc/kubernetes/manifests/kube-apiserver.yaml /kube-
apiserver-yaml-backup // create a backup incase we break anything
docker@minikube:~$ sudo vi /etc/kubernetes/manifests/kube-apiserver.yaml // minikube
doesn't have vim
```

Add AlwaysPullImages, in the list, exit. The API server will restart, this may take about 30 seconds.



```
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.49.2
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/var/lib/minikube/certs/ca.crt
    - --enable-admission-plugins=AlwaysPullImages,NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTolerationSeconds,
NodeRestriction,MutatingAdmissionWebhook,ValidatingAdmissionWebhook,ResourceQuota
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/var/lib/minikube/certs/etcd/ca.crt
    - --etcd-certfile=/var/lib/minikube/certs/apiserver-etcd-client.crt
    - --etcd-keyfile=/var/lib/minikube/certs/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
```

```
$ kubectl get po -A // Wait until you see all pods are READY
NAMESPACE NAME READY STATUS RESTARTS AGE
default admitted 1/1 Running 0 8m43s
kube-system calico-kube-controllers-558d465845-6kxrt 1/1 Running 5 (4h57m ago) 29h
kube-system calico-node-2v2tb 1/1 Running 3 (4h57m ago) 18h
kube-system calico-node-7mglq 1/1 Running 4 (4h57m ago) 29h
kube-system calico-node-9ch77 1/1 Running 1 (4h57m ago) 14h
kube-system coredns-5dd5756b68-br7nd 1/1 Running 5 (4h57m ago) 29h
kube-system etcd-minikube 1/1 Running 4 (4h57m ago) 29h
kube-system kube-apiserver-minikube 1/1 Running 1 90s
kube-system kube-controller-manager-minikube 1/1 Running 4 (4h57m ago) 29h
kube-system kube-proxy-d96lm 1/1 Running 3 (4h57m ago) 18h
kube-system kube-proxy-l5xrl 1/1 Running 1 (4h57m ago) 14h
kube-system kube-proxy-vr5n5 1/1 Running 4 (4h57m ago) 29h
kube-system kube-scheduler-minikube 1/1 Running 4 (4h57m ago) 29h
kube-system storage-provisioner 1/1 Running 12 (111s ago) 29h
kubernetes-dashboard dashboard-metrics-scraper-7fd5cb4ddc-d6ts8 1/1 Running 5 (4h57m
ago) 29h
kubernetes-dashboard kubernetes-dashboard-8694d4445c-xwdtb 1/1 Running 6 (4h57m ago)
29h
ns-for-demo nginx-b4ccb96c6-hxhnf 1/1 Running 0 95m

$ kubectl describe pod kube-apiserver-minikube -n kube-system | grep -i admission //
Validate the AlwaysPullImages Controller is running in API Server
--enable-admission-
plugins=AlwaysPullImages,NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorag
eClass,DefaultTolerationSeconds,NodeRestriction,MutatingAdmissionWebhook,ValidatingA
dmissionWebhook,ResourceQuota

$ kubectl run mutated --image=nginx --image-pull-policy=IfNotPresent // Create
another pod, named is "mutated" but everything is the same with admitted pod
pod/mutated created
$ kubectl get pod mutated -o yaml | grep -i imagepull // The controller is force the
imagePullPolicy to be "Always"
imagePullPolicy: Always
```

```
One interesting thing is, the admitted pod doesn't not affected by the new manifest
$ kubectl get pod admitted -o yaml | grep -i imagepull
imagePullPolicy: IfNotPresent
```