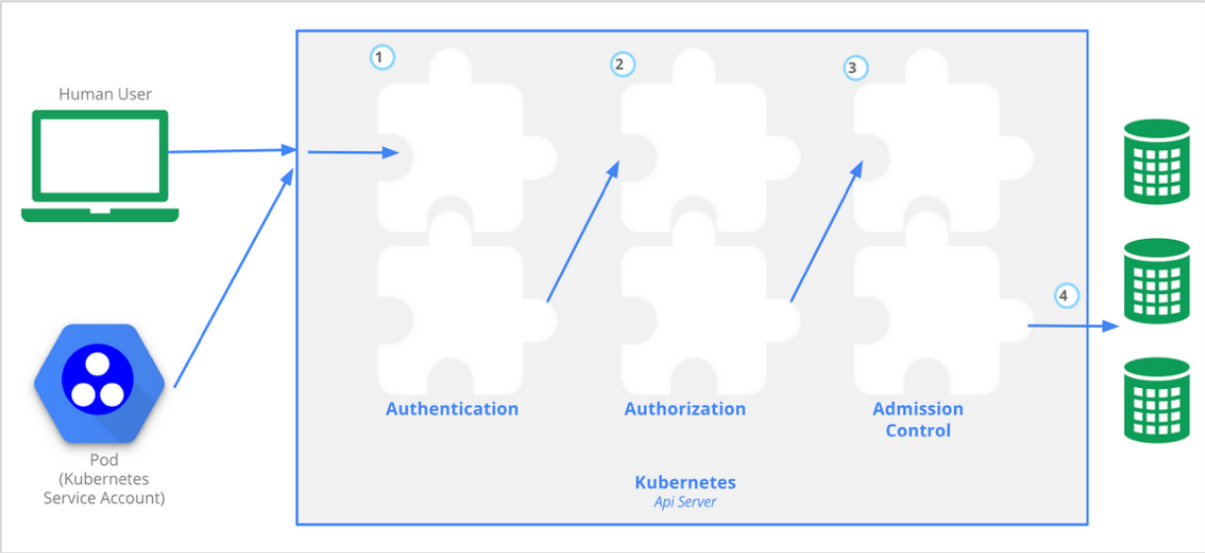
Chapter 9 Authentication, Authorization, Admission Control

- Each access request goes through 3 stages:



  - Authentication: logs in a user
  - Authorization: authorizes the API requests
  - Admission Control: modify or reject the requests based on some additional checks (i.e. Quota)

Authentication

- Two kinds of users
  - Normal users
    - managed outside of K8s cluster via independent services
      - i.e. user/client certificates, file listing usernames/passwords, google accounts, etc.
  - Service accounts
    - usually created automatically via API server
    - tied to a given Namespace
    - mount the creds to communicate with the API server as Secrets
- Authentication modules:
  - Client Certificates `--client-ca-file=SOMEFILE`
  - Static Token File `--token-auth-file=SOMEFILE`
    - pre-defined bearer tokens
  - Bootstrap Tokens
    - bootstrapping a new K8s cluster
  - Static Password File `--basic-auth-file=SOMEFILE`
    - passwords cannot be changed without restarting the API server

    - Service Account Tokens
        - automatically enabled authenticator
        - uses signed bearer tokens to verify requests
        - tokens get attached to Pods using the ServiceAccount Admission Controller
            - allows in-cluster processes to talk to the Api server
    - OpenID Connect Tokens
        - connects with OAuth2 providers to offload the auth to external services
    - Webhook Token Authentication
        - verification of bearer tokens can be offloaded to a remote service
    - Authenticating Proxy
- can enable multiple authenticators
    - usually service account tokens authenticator & user authenticator

Authorization modules

- Node Authorizer
    - authorizes kubelet's read requests for services, endpoints, nodes, etc. and write requests for nodes, pods, events, etc.
- Attribute-Based Access Control (ABAC) Authorizer
    - combines policies with attributes

```
{
  "apiVersion": "abac.authorization.kubernetes.io/v1beta1",
  "kind": "Policy",
  "spec": {
    "user": "student",
    "namespace": "lfs158",
    "resource": "pods",
    "readonly": true
  }
}
```

    - user `student` can only read Pods in the Namespace `lfs158`
    - to enable, start the API server with `--authorization-mode=ABAC` and `--authorization-policy-file=PolicyFile.json`
- Webhook Authorizer
    - offer authorization decisions to 3rd party services that returns true/false
    - `--authorization-webhook-config-file=SOME_FILENAME`
- Role-Based Access Control (RBAC) Authorizer

- `--authorization-mode=RBAC`
- regulate based on roles of users
- restrict resource access by operations (create, get, update, etc.)
- Two kinds of roles:
  - Role: grant access to resources within a specific Namespace
  - ClusterRole: similar to Role, scope is cluster-wide

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: lfs158
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

- `pod-reader` role has access to `lfs158` NS
- Once the role is created, bind users with RoleBinding
- Two kinds of RoleBindings
  - RoleBinding
    - bind users to the same namespace as a Role/ClusterRole
  - ClusterRoleBinding
    - RoleBinding at cluster-level and to all NameSpaces

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: pod-read-access
  namespace: lfs158
subjects:
- kind: User
  name: student
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

- `student` user can read the Pods of `lfs158` NS

Admission Control

- Specify granular access control policies
- force policies using different admission controllers (plugins)
- start the API server with `--enable-admission-plugins=NamespaceLifecycle,ResourceQuota,PodSecurityPolicy,DefaultStorageClass`

Demo

LINIKLFS2017-V001100_D...
Aug 18, 2019