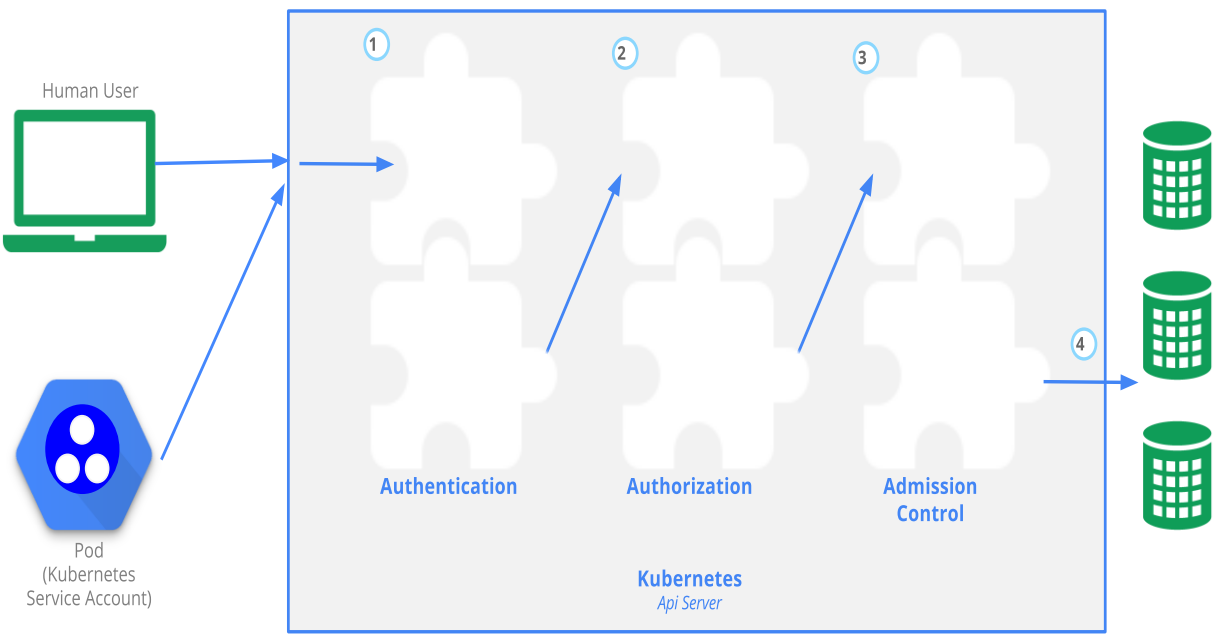
Kubernetes API Server

When creating a resource from a JSON file, for example, `kubectl` posts the file's contents to the API server through an HTTP POST request. In the API server, it follows the following steps.



1. AUTHENTICATING THE CLIENT WITH AUTHENTICATION PLUGINS (Auth N)

One or more authentication plugins configured in the API server.

(The authentication plugins are enabled through command-line options when startingthe API server)

Determines who is sending the request. It does this by inspecting the HTTP request.

Depending on the authentication method, the user info. can be extracted from the client's certificate or an HTTP header.

This data is then used in the next stage, which is authorization.

2. AUTHORIZING THE CLIENT WITH AUTHORIZATION PLUGINS (Auth Z)

API server is also configured to use one or more authorization plugins. Their job is **to determine whether the authenticated user can perform the requested action on the requested resource.**

For example, when creating pods, the API server consults all authorization plugins in turn, to determine whetherthe user can create pods in the requested namespace. As soon as a plugin says the user can perform the action, the API server progresses to the next stage.

3. VALIDATING AND/OR MODIFYING THE RESOURCE IN THE REQUEST WITH ADMISSION CONTROL PLUGINS

If the request is trying to create, modify, or delete a resource, the request is sent through Admission Control. Again, the server is configured with multiple Admission Control plugins. **These plugins can modify the resource for different reasons.** They may initialize fields missing from the resource specification to the configured default values or even override them. They may even modify other related resources, which aren't in the request, and can also reject a request for whatever reason. The resourcepasses through all Admission Control plugins.

NOTE When the request is only trying to read data, the request doesn't gothrough the Admission Control.

Examples of Admission Control plugins

- `AlwaysPullImages`—Overrides the pod's `imagePullPolicy` to `Always`, forcing the image to be pulled every time the pod is deployed.
- `ServiceAccount`—Applies the default service account to pods that don't specify it explicitly.
- `NamespaceLifecycle`—Prevents creation of pods in namespaces that are in the process of being deleted, as well as in non-existing namespaces.
- `ResourceQuota`—Ensures pods in a certain namespace only use as much CPU and memory as has been allotted to the namespace. We'll learn more about this in chapter 14.

** a list of additional Admission Control plugin at https://kubernetes.io/docs/admin/admission-controllers/.

4. VALIDATING THE RESOURCE AND STORING IT PERSISTENTLY

After letting the request pass through all the Admission Control plugins, the API server then validates the object, stores it in etcd, and returns a response to the client

Authentication

Two kinds of clients connecting to the API server:

- Actual humans (users)
- Pods (applications running inside them)
-

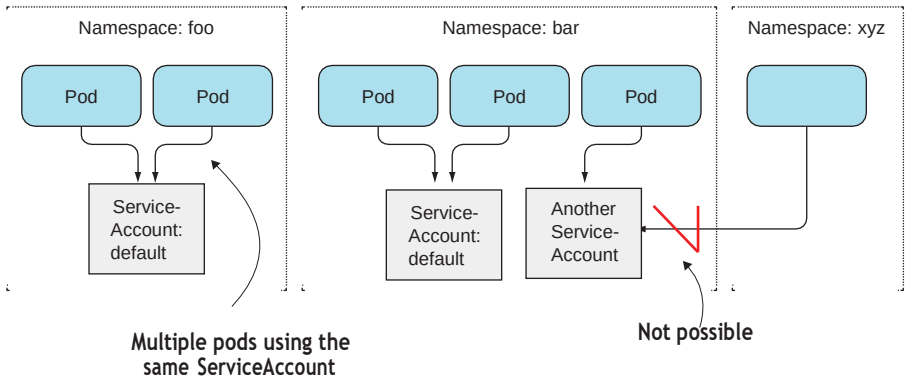
- **Users are meant to be managed by an external system**, such as a Single SignOn (SSO) system, and No resource relate to users.
- **The pods use the *service accounts*, which are created and stored in the cluster as ServiceAccount resources**.

Both human users and ServiceAccounts can belong to one or more groups.

The authentication plugin returns groups along with the username and user ID.

- **Service Account**

  Each pod associated with a single ServiceAccount in the pod's namespace

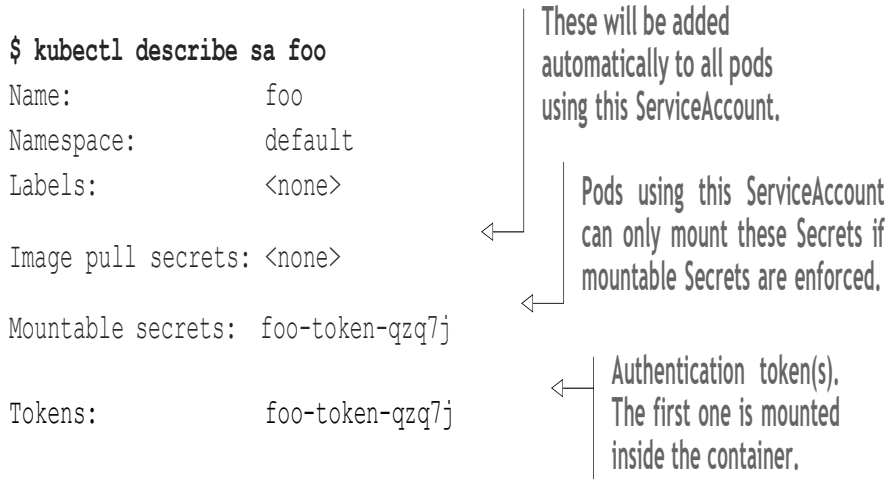


Assign a ServiceAccount to a pod by specifying the account's name in the pod manifest. (Not assigned, then **Default Service Account** in the namespace)

By assigning different ServiceAccounts to pods, you can **control which resources each pod has access to**. When a request bearing the authentication token is received by the API server, the server uses the token to authenticate the client sending the request and then determines **whether or not the related ServiceAccount is allowed to perform the requested operation**. The API server obtains this information from the system-wide authorization plugin configured by the cluster administrator. One of the available authorization plugins is **the role-based access control (RBAC) plugin.**

$ kubectl create serviceaccount foo

serviceaccount "foo" created



The SERVICE ACCOUNT'S TOKEN is used to talk to the API server.

By setting the name of the ServiceAccount in the spec.serviceAccountName field in the pod definition, the service account created is assigned to pods.

Authorization

An authorization plugin such as **RBAC**(Role Based Access Control), which runs inside the API server, determines whether a client is allowed to perform the requested verb on the requested resource or not.

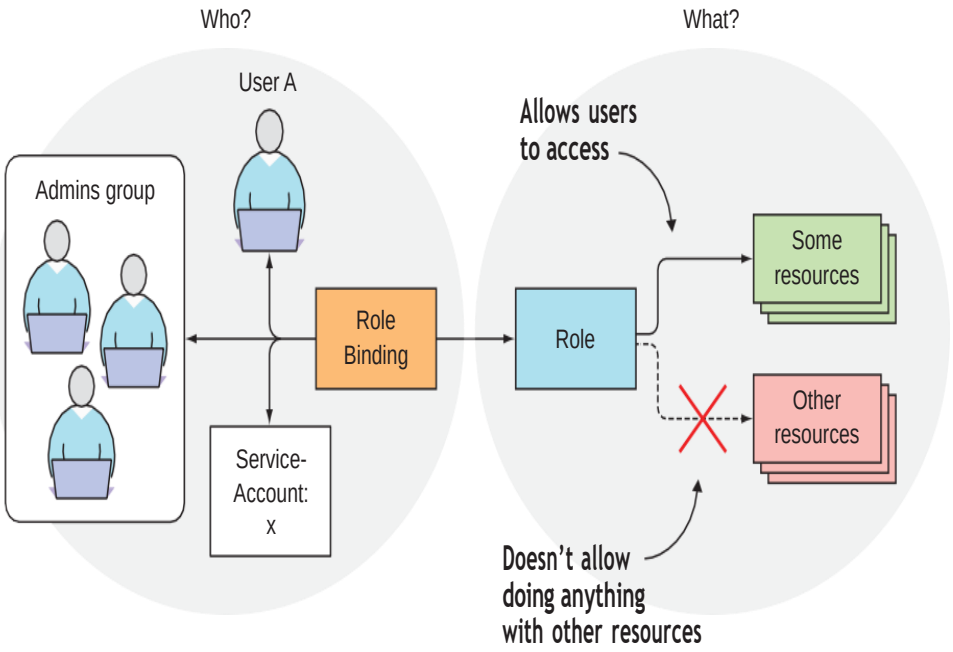
HTTP method	Verb for single resource	Verb for collection
GET, HEAD	get (and watch for watching)	list (and watch)
POST	create	n/a
PUT	update	n/a
PATCH	patch	n/a
DELETE	delete	deletecollection

The RBAC authorization rules are configured through four resources, which can be grouped into two groups:

Roles and ClusterRoles, which specify which verbs can be performed on which resources.

RoleBindings and ClusterRoleBindings, which bind the above roles to specific users, groups, or ServiceAccounts.

Roles define **what can be done**,
Bindings define **who can do it**



Ex: A definition of a `Role`: service-reader.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace:  foo
  name: service-reader
rules:
- apiGroups: [""]
  verbs: ["get", "list"]
  resources: ["services"]
```

Roles are namespaced (if namespace is omitted, the current namespace is used).

Services are resources in the core apiGroup, which has no name - hence the "".

Getting individual Services (by name) and listing all of them is allowed.

This rule pertains to services (plural name must be used!).

WARNING The plural form must be used when specifying resources