Using ABAC Authorization

By Authorization Algorithm

Archived: 2026-04-06 01:54:57 UTC

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.

Policy File Format

To enable `ABAC` mode, specify `--authorization-policy-file=SOME_FILENAME` and `--authorization-mode=ABAC` on startup.

The file format is [one JSON object per line](). There should be no enclosing list or map, only one map per line.

Each line is a "policy object", where each such object is a map with the following properties:

- Versioning properties:
  - `apiVersion`, type string; valid values are "abac.authorization.kubernetes.io/v1beta1". Allows versioning and conversion of the policy format.
  - `kind`, type string: valid values are "Policy". Allows versioning and conversion of the policy format.
- `spec` property set to a map with the following properties:
  - Subject-matching properties:
    - `user`, type string; the user-string from `--token-auth-file`. If you specify `user`, it must match the username of the authenticated user.
    - `group`, type string; if you specify `group`, it must match one of the groups of the authenticated user. `system:authenticated` matches all authenticated requests. `system:unauthenticated` matches all unauthenticated requests.
  - Resource-matching properties:
    - `apiGroup`, type string; an API group.
      - Ex: `apps`, `networking.k8s.io`
      - Wildcard: `*` matches all API groups.
    - `namespace`, type string; a namespace.
      - Ex: `kube-system`
      - Wildcard: `*` matches all resource requests.
    - `resource`, type string; a resource type
      - Ex: `pods`, `deployments`
      - Wildcard: `*` matches all resource requests.
  - Non-resource-matching properties:
    - `nonResourcePath`, type string; non-resource request paths.
      - Ex: `/version` or `/apis`

        - Wildcard:
            - `*` matches all non-resource requests.
            - `/foo/*` matches all subpaths of `/foo/`.
    - `readonly`, type boolean, when true, means that the Resource-matching policy only applies to get, list, and watch operations, Non-resource-matching policy only applies to get operation.

Note:

An unset property is the same as a property set to the zero value for its type (e.g. empty string, 0, false). However, unset should be preferred for readability.

In the future, policies may be expressed in a JSON format, and managed via a REST interface.

A request has attributes which correspond to the properties of a policy object.

When a request is received, the attributes are determined. Unknown attributes are set to the zero value of its type (e.g. empty string, 0, false).

A property set to `"*"` will match any value of the corresponding attribute.

The tuple of attributes is checked for a match against every policy in the policy file. If at least one line matches the request attributes, then the request is authorized (but may fail later validation).

To permit any authenticated user to do something, write a policy with the group property set to `"system:authenticated"`.

To permit any unauthenticated user to do something, write a policy with the group property set to `"system:unauthenticated"`.

To permit a user to do anything, write a policy with the apiGroup, namespace, resource, and nonResourcePath properties set to `"*"`.

Kubectl

Kubectl uses the `/api` and `/apis` endpoints of apiserver to discover served resource types, and validates objects sent to the API by create/update operations using schema information located at `/openapi/v2`.

When using ABAC authorization, those special resources have to be explicitly exposed via the `nonResourcePath` property in a policy (see examples below):

- `/api`, `/api/*`, `/apis`, and `/apis/*` for API version negotiation.
- `/version` for retrieving the server version via `kubectl version`.
- `/swaggerapi/*` for create/update operations.

To inspect the HTTP calls involved in a specific kubectl operation you can turn up the verbosity:

```
kubectl --v=8 version
```

Examples

1. Alice can do anything to all resources:

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "alice",
```

2. The kubelet can read any pods:

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet"
```

3. The kubelet can read and write events:

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet"
```

4. Bob can just read pods in namespace "projectCaribou":

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "bob", "na
```

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

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"group": "system:a
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"group": "system:u
```

Complete file example

A quick note on service accounts

Every service account has a corresponding ABAC username, and that service account's username is generated according to the naming convention:

```
system:serviceaccount:<namespace>:<serviceaccountname>
```

Creating a new namespace leads to the creation of a new service account in the following format:

```
system:serviceaccount:<namespace>:default
```

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
```

The apiserver will need to be restarted to pick up the new policy lines.

Last modified February 18, 2024 at 10:07 AM PST: Reorder authn/authz pages (9f327512c6)
