Configure Service Accounts for Pods

Archived: 2026-04-05 22:25:33 UTC

Kubernetes offers two distinct ways for clients that run within your cluster, or that otherwise have a relationship to your cluster's control plane to authenticate to the API server.

A *service account* provides an identity for processes that run in a Pod, and maps to a ServiceAccount object. When you authenticate to the API server, you identify yourself as a particular *user*. Kubernetes recognises the concept of a user, however, Kubernetes itself does **not** have a User API.

This task guide is about ServiceAccounts, which do exist in the Kubernetes API. The guide shows you some ways to configure ServiceAccounts for Pods.

Before you begin

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using minikube or you can use one of these Kubernetes playgrounds:

- iximiuz Labs
- Killercoda
- KodeKloud

Use the default service account to access the API server

When Pods contact the API server, Pods authenticate as a particular ServiceAccount (for example, `default`). There is always at least one ServiceAccount in each namespace.

Every Kubernetes namespace contains at least one ServiceAccount: the default ServiceAccount for that namespace, named `default`. If you do not specify a ServiceAccount when you create a Pod, Kubernetes automatically assigns the ServiceAccount named `default` in that namespace.

You can fetch the details for a Pod you have created. For example:

```
kubectl get pods/<podname> -o yaml
```

In the output, you see a field `spec.serviceAccountName`. Kubernetes automatically sets that value if you don't specify it when you create a Pod.

An application running inside a Pod can access the Kubernetes API using automatically mounted service account credentials. See accessing the Cluster to learn more.

When a Pod authenticates as a ServiceAccount, its level of access depends on the authorization plugin and policy in use.

The API credentials are automatically revoked when the Pod is deleted, even if finalizers are in place. In particular, the API credentials are revoked 60 seconds beyond the `.metadata.deletionTimestamp` set on the Pod (the deletion timestamp is typically the time that the **delete** request was accepted plus the Pod's termination grace period).

Opt out of API credential automounting

If you don't want the kubelet to automatically mount a ServiceAccount's API credentials, you can opt out of the default behavior. You can opt out of automounting API credentials on `/var/run/secrets/kubernetes.io/serviceaccount/token` for a service account by setting `automountServiceAccountToken: false` on the ServiceAccount:

For example:

```
 apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
automountServiceAccountToken: false
...
```

You can also opt out of automounting API credentials for a particular Pod:

```
 apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  serviceAccountName: build-robot
  automountServiceAccountToken: false
  ...
```

If both the ServiceAccount and the Pod's `.spec` specify a value for `automountServiceAccountToken`, the Pod spec takes precedence.

Use more than one ServiceAccount

Every namespace has at least one ServiceAccount: the default ServiceAccount resource, called `default`. You can list all ServiceAccount resources in your current namespace with:

```
kubectl get serviceaccounts
```

The output is similar to this:

```
 NAME      SECRETS    AGE
default   1          1d
```

You can create additional ServiceAccount objects like this:

```
 kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
EOF
```

The name of a ServiceAccount object must be a valid DNS subdomain name.

If you get a complete dump of the service account object, like this:

```
 kubectl get serviceaccounts/build-robot -o yaml
```

The output is similar to this:

```
 apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: 2019-06-16T00:12:34Z
  name: build-robot
  namespace: default
  resourceVersion: "272500"
  uid: 721ab723-13bc-11e5-aec2-42010af0021e
```

You can use authorization plugins to set permissions on service accounts.

To use a non-default service account, set the `spec.serviceAccountName` field of a Pod to the name of the ServiceAccount you wish to use.

You can only set the `serviceAccountName` field when creating a Pod, or in a template for a new Pod. You cannot update the `.spec.serviceAccountName` field of a Pod that already exists.

Note:

The `.spec.serviceAccount` field is a deprecated alias for `.spec.serviceAccountName`. If you want to remove the fields from a workload resource, set both fields to empty explicitly on the pod template.

Cleanup

If you tried creating `build-robot` ServiceAccount from the example above, you can clean it up by running:

```
kubectl delete serviceaccount/build-robot
```

Manually create an API token for a ServiceAccount

Suppose you have an existing service account named "build-robot" as mentioned earlier.

You can get a time-limited API token for that ServiceAccount using `kubectl`:

```
kubectl create token build-robot
```

The output from that command is a token that you can use to authenticate as that ServiceAccount. You can request a specific token duration using the `--duration` command line argument to `kubectl create token` (the actual duration of the issued token might be shorter, or could even be longer).

FEATURE STATE: `Kubernetes v1.33 [stable]` (enabled by default)

Using `kubectl` v1.31 or later, it is possible to create a service account token that is directly bound to a Node:

```
kubectl create token build-robot --bound-object-kind Node --bound-object-name node-001 --bound-object-uid 123.
```

The token will be valid until it expires or either the associated Node or service account are deleted.

Note:

Versions of Kubernetes before v1.22 automatically created long term credentials for accessing the Kubernetes API. This older mechanism was based on creating token Secrets that could then be mounted into running Pods. In more recent versions, including Kubernetes v1.35, API credentials are obtained directly by using the TokenRequest API, and are mounted into Pods using a projected volume. The tokens obtained using this method have bounded lifetimes, and are automatically invalidated when the Pod they are mounted into is deleted.

You can still manually create a service account token Secret; for example, if you need a token that never expires. However, using the TokenRequest subresource to obtain a token to access the API is recommended instead.

Manually create a long-lived API token for a ServiceAccount

If you want to obtain an API token for a ServiceAccount, you create a new Secret with a special annotation, `kubernetes.io/service-account.name`.

```
 kubectl apply -f - <<EOF
apiVersion: v1
```

```
kind: Secret
metadata:
  name: build-robot-secret
  annotations:
    kubernetes.io/service-account.name: build-robot
type: kubernetes.io/service-account-token
EOF
```

If you view the Secret using:

```
kubectl get secret/build-robot-secret -o yaml
```

you can see that the Secret now contains an API token for the "build-robot" ServiceAccount.

Because of the annotation you set, the control plane automatically generates a token for that ServiceAccounts, and stores them into the associated Secret. The control plane also cleans up tokens for deleted ServiceAccounts.

```
kubectl describe secrets/build-robot-secret
```

The output is similar to this:

```
 Name:           build-robot-secret
Namespace:      default
Labels:         <none>
Annotations:    kubernetes.io/service-account.name: build-robot
                kubernetes.io/service-account.uid: da68f9c6-9d26-11e7-b84e-002dc52800da

Type:   kubernetes.io/service-account-token

Data
====
ca.crt:         1338 bytes
namespace:      7 bytes
token:          ...
```

Note:

The content of `token` is omitted here.

Take care not to display the contents of a `kubernetes.io/service-account-token` Secret somewhere that your terminal / computer screen could be seen by an onlooker.

When you delete a ServiceAccount that has an associated Secret, the Kubernetes control plane automatically cleans up the long-lived token from that Secret.

Note:

If you view the ServiceAccount using:

```
kubectl get serviceaccount build-robot -o yaml
```

You can't see the `build-robot-secret` Secret in the ServiceAccount API objects `.secrets` field because that field is only populated with auto-generated Secrets.

Add ImagePullSecrets to a service account

First, create an imagePullSecret. Next, verify it has been created. For example:

- Create an imagePullSecret, as described in Specifying ImagePullSecrets on a Pod.

  ```
  kubectl create secret docker-registry myregistrykey --docker-server=<registry name> \
         --docker-username=DUMMY_USERNAME --docker-password=DUMMY_DOCKER_PASSWORD \
         --docker-email=DUMMY_DOCKER_EMAIL
  ```

- Verify it has been created.

  ```
  kubectl get secrets myregistrykey
  ```

  The output is similar to this:

  ```
  NAME             TYPE                              DATA    AGE
  myregistrykey    kubernetes.io/.dockerconfigjson   1       1d
  ```

Add image pull secret to service account

Next, modify the default service account for the namespace to use this Secret as an imagePullSecret.

```
kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "myregistrykey"}]}'
```

You can achieve the same outcome by editing the object manually:

```
kubectl edit serviceaccount/default
```

The output of the `sa.yaml` file is similar to this:

Your selected text editor will open with a configuration looking something like this:

```
 apiVersion: v1
kind: ServiceAccount
```

```
metadata:
  creationTimestamp: 2021-07-07T22:02:39Z
  name: default
  namespace: default
  resourceVersion: "243024"
  uid: 052fb0f4-3d50-11e5-b066-42010af0d7b6
```

Using your editor, delete the line with key `resourceVersion`, add lines for `imagePullSecrets:` and save it. Leave the `uid` value set the same as you found it.

After you made those changes, the edited ServiceAccount looks something like this:

```
 apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: 2021-07-07T22:02:39Z
  name: default
  namespace: default
  uid: 052fb0f4-3d50-11e5-b066-42010af0d7b6
imagePullSecrets:
  - name: myregistrykey
```

Verify that imagePullSecrets are set for new Pods

Now, when a new Pod is created in the current namespace and using the default ServiceAccount, the new Pod has its `spec.imagePullSecrets` field set automatically:

```
 kubectl run nginx --image=<registry name>/nginx --restart=Never
kubectl get pod nginx -o=jsonpath='{.spec.imagePullSecrets[0].name}{"\n"}'
```

The output is:

```
 myregistrykey
```

ServiceAccount token volume projection

FEATURE STATE: `Kubernetes v1.20 [stable]`

Note:

To enable and use token request projection, you must specify each of the following command line arguments to `kube-apiserver`:

`--service-account-issuer`

defines the Identifier of the service account token issuer. You can specify the `--service-account-issuer` argument multiple times, this can be useful to enable a non-disruptive change of the issuer. When this flag is specified multiple times, the first is used to generate tokens and all are used to determine which issuers are accepted. You must be running Kubernetes v1.22 or later to be able to specify `--service-account-issuer` multiple times.

`--service-account-key-file`
: specifies the path to a file containing PEM-encoded X.509 private or public keys (RSA or ECDSA), used to verify ServiceAccount tokens. The specified file can contain multiple keys, and the flag can be specified multiple times with different files. If specified multiple times, tokens signed by any of the specified keys are considered valid by the Kubernetes API server.

`--service-account-signing-key-file`
: specifies the path to a file that contains the current private key of the service account token issuer. The issuer signs issued ID tokens with this private key.

`--api-audiences` (can be omitted)
: defines audiences for ServiceAccount tokens. The service account token authenticator validates that tokens used against the API are bound to at least one of these audiences. If `api-audiences` is specified multiple times, tokens for any of the specified audiences are considered valid by the Kubernetes API server. If you specify the `--service-account-issuer` command line argument but you don't set `--api-audiences`, the control plane defaults to a single element audience list that contains only the issuer URL.

The kubelet can also project a ServiceAccount token into a Pod. You can specify desired properties of the token, such as the audience and the validity duration. These properties are *not* configurable on the default ServiceAccount token. The token will also become invalid against the API when either the Pod or the ServiceAccount is deleted.

You can configure this behavior for the `spec` of a Pod using a projected volume type called `ServiceAccountToken`.

The token from this projected volume is a JSON Web Token (JWT). The JSON payload of this token follows a well defined schema - an example payload for a pod bound token:

```
{
 "aud": [  # matches the requested audiences, or the API server's default audiences when none are explicitly re
   "https://kubernetes.default.svc"
 ],
 "exp": 1731613413,
 "iat": 1700077413,
 "iss": "https://kubernetes.default.svc",  # matches the first value passed to the --service-account-issuer fla
 "jti": "ea28ed49-2e11-4280-9ec5-bc3d1d84661a",
 "kubernetes.io": {
   "namespace": "kube-system",
   "node": {
     "name": "127.0.0.1",
     "uid": "58456cb0-dd00-45ed-b797-5578fdceaced"
   },
```

```
    "pod": {
      "name": "coredns-69cbfb9798-jv9gn",
      "uid": "778a530c-b3f4-47c0-9cd5-ab018fb64f33"
    },
    "serviceaccount": {
      "name": "coredns",
      "uid": "a087d5a0-e1dd-43ec-93ac-f13d89cd13af"
    },
    "warnafter": 1700081020
  },
  "nbf": 1700077413,
  "sub": "system:serviceaccount:kube-system:coredns"
}
```

Launch a Pod using service account token projection

To provide a Pod with a token with an audience of `vault` and a validity duration of two hours, you could define a Pod manifest that is similar to:

```
 apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    volumeMounts:
    - mountPath: /var/run/secrets/tokens
      name: vault-token
  serviceAccountName: build-robot
  volumes:
  - name: vault-token
    projected:
      sources:
      - serviceAccountToken:
          path: vault-token
          expirationSeconds: 7200
          audience: vault
```

Create the Pod:

```
kubectl create -f https://k8s.io/examples/pods/pod-projected-svc-token.yaml
```

The kubelet will: request and store the token on behalf of the Pod; make the token available to the Pod at a configurable file path; and refresh the token as it approaches expiration. The kubelet proactively requests rotation for the token if it is older than 80% of its total time-to-live (TTL), or if the token is older than 24 hours.

The application is responsible for reloading the token when it rotates. It's often good enough for the application to load the token on a schedule (for example: once every 5 minutes), without tracking the actual expiry time.

Service account issuer discovery

FEATURE STATE: `Kubernetes v1.21 [stable]`

If you have enabled token projection for ServiceAccounts in your cluster, then you can also make use of the discovery feature. Kubernetes provides a way for clients to federate as an *identity provider*, so that one or more external systems can act as a *relying party*.

Note:

The issuer URL must comply with the OIDC Discovery Spec. In practice, this means it must use the `https` scheme, and should serve an OpenID provider configuration at `{service-account-issuer}/.well-known/openid-configuration`.

If the URL does not comply, ServiceAccount issuer discovery endpoints are not registered or accessible.

When enabled, the Kubernetes API server publishes an OpenID Provider Configuration document via HTTP. The configuration document is published at `/.well-known/openid-configuration`. The OpenID Provider Configuration is sometimes referred to as the *discovery document*. The Kubernetes API server publishes the related JSON Web Key Set (JWKS), also via HTTP, at `/openid/v1/jwks`.

Note:

The responses served at `/.well-known/openid-configuration` and `/openid/v1/jwks` are designed to be OIDC compatible, but not strictly OIDC compliant. Those documents contain only the parameters necessary to perform validation of Kubernetes service account tokens.

Clusters that use RBAC include a default ClusterRole called `system:service-account-issuer-discovery`. A default ClusterRoleBinding assigns this role to the `system:serviceaccounts` group, which all ServiceAccounts implicitly belong to. This allows pods running on the cluster to access the service account discovery document via their mounted service account token. Administrators may, additionally, choose to bind the role to `system:authenticated` or `system:unauthenticated` depending on their security requirements and which external systems they intend to federate with.

The JWKS response contains public keys that a relying party can use to validate the Kubernetes service account tokens. Relying parties first query for the OpenID Provider Configuration, and use the `jwks_uri` field in the response to find the JWKS.

In many cases, Kubernetes API servers are not available on the public internet, but public endpoints that serve cached responses from the API server can be made available by users or by service providers. In these cases, it is possible to override the `jwks_uri` in the OpenID Provider Configuration so that it points to the public endpoint, rather than the API server's address, by passing the `--service-account-jwks-uri` flag to the API server. Like the issuer URL, the JWKS URI is required to use the `https` scheme.

What's next

See also:

- Read the Cluster Admin Guide to Service Accounts
- Read about Authorization in Kubernetes
- Read about Secrets
    - or learn to distribute credentials securely using Secrets
    - but also bear in mind that using Secrets for authenticating as a ServiceAccount is deprecated. The recommended alternative is ServiceAccount token volume projection.
- Read about projected volumes.
- For background on OIDC discovery, read the ServiceAccount signing key retrieval Kubernetes Enhancement Proposal
- Read the OIDC Discovery Spec
