Keycloak Federated Client Authentication

No More Secrets With Keycloak's Federated Client Authentication

OBJECTIVE

Demonstrate Keycloak's new federated client authentication feature, how it works with Kubernetes and SPIFFE, and why it matters for cloud-native deployments.

Identity and Access Management with Keycloak

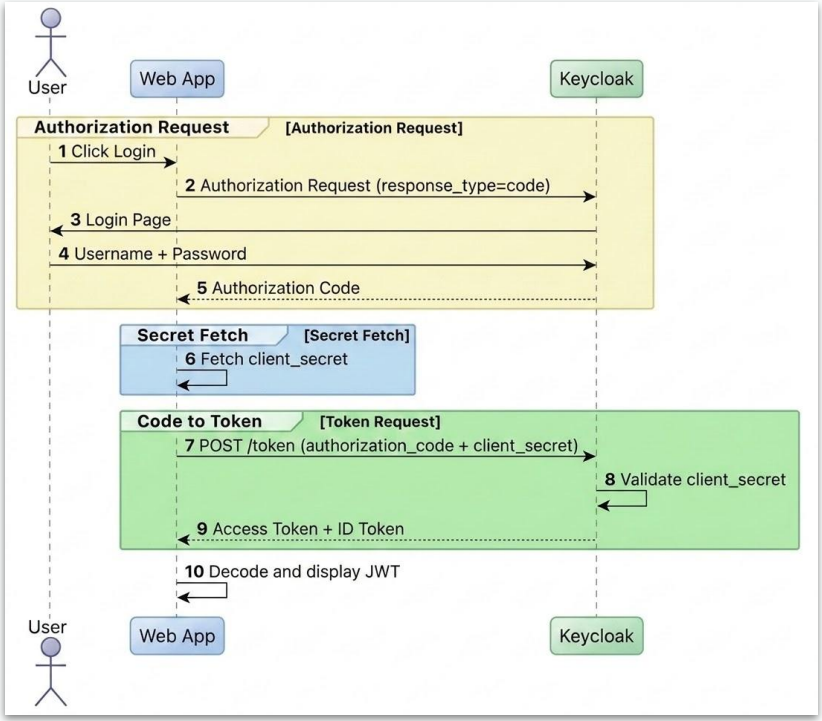
Core Standards

- Implements foundational standards: OAuth 2.0, SAML and OpenID Connect (OIDC).
- Acts as a centralized server for authentication, enabling Single Sign-On (SSO), user management, and authorization, allowing developers to avoid writing custom security code.

CNCF Ecosystem Role

- Centralized Identity: Applications delegate all user authentication and SSO chores.
- API Security Plane: Issues access tokens for securing microservices.
- Manages complex user flows: MFA and external Identity Federation.

Authorization Code Flow: User Authentication



Client Credential Flow: Applications Acting on Their Own Behalf

Static Secrets Fail at Scale

Traditional Client Authentication

- In **Keycloak** Client secrets are stored persistently.
- Credentials must be **distributed** to **every client instance** (ConfigMaps).
- **Manual rotation** across all deployments introduces high coordination risk.
- **Security Risk:** Static credential leakage grants persistent access.

SOLUTION

Cloud-Native Challenge & Integration

- Hundreds of microservices require dedicated OAuth clients.
- Kubernetes and SPIFFE already issue trusted, short-lived workload identity.

The Solution: Keycloak validates platform-issued tokens instead of managing its own secrets.

Workload Identity: The Path to Secretless Clients

- Keycloak 26.6 introduces **Federated Client Authentication**.
- OIDC clients authenticate using **ephemeral tokens**.
- Native integration with **Kubernetes** and **SPIFFE** identities.
- Validates platform identity—eliminating stored client secrets.

Federated Client Authentication

Keycloak and Platform Engineering Benefits

- **Zero Secret Management:** Completely eliminates storing client secrets.
- **Automatic Rotation:** Token lifecycle is inherently handled by the platform.
- **Enhanced Security:** Credentials are short-lived and cryptographically signed.
- **Operational Consistency:** One identity model across all service meshes/clouds.

The Outcome: A unified identity plane, not a segregated secrets store.

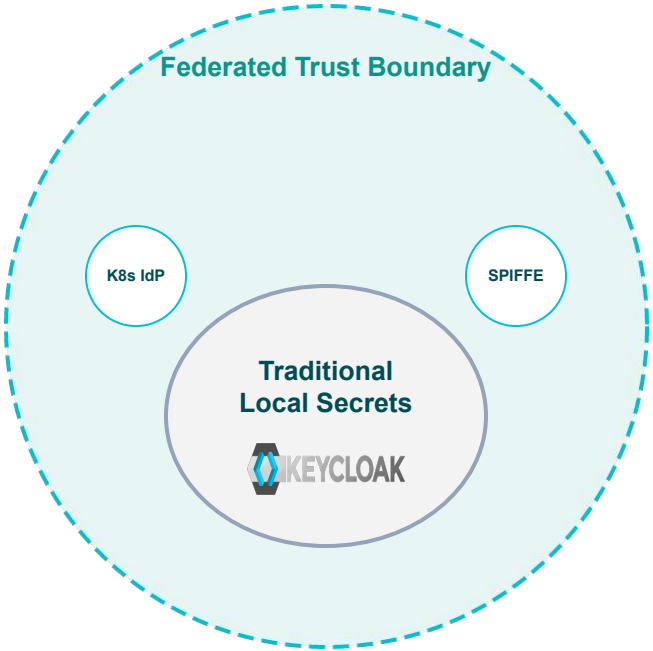
Keycloak's Client Authentication Methods

Client Authentication Capability Matrix

Authenticator Type	How it Works	Primary Use Case
Client ID and Secret	Simple Shared Secret	Legacy applications, non-containerized workloads.
Signed JWT	Client Signs Own JWT	Securing clients who can generate and rotate their own keys.
Signed JWT - Federated	**External IdP JWT Validation**	**Cloud-native workloads (K8s Service Accounts, SPIFFE).**
x509 Certificate	Mutual TLS (mTLS)	Certificate-based authentication, common in older enterprise environments.

- **Starting Keycloak 26.6: Signed JWT - Federated** enables validating tokens from **Kubernetes** and **SPIFFE**.

The Shifting Trust Boundary: From Local Secrets to External Identity



- **Traditional Trust:** Boundary defined by local client metadata and stored secrets.
- **The Federation:** Trust boundary extends to authoritative external Issuers (Kubernetes and SPIFFE).
- **Keycloak's New Role:** Accepts external workload identity tokens (JWTs) as proof of client identity.
- **Outcome:** Delegation of identity validation enables Federated Client Authentication.

Keycloak's Federated Identity Topology

- **Dedicated Realm:** A focused **kubernetes** Realm isolates workload identities.
- **Identity Provider:** IdP configured to demonstrate authentication through Kubernetes OIDC.
- **Federated Clients:** Two clients are provisioned for testing: one using federated-jwt and one using a traditional secret for comparison.

```
Realm: kubernetes
Identity Providers:
├─ kubernetes (Issuer: https://kubernetes.default.svc.cluster.local)
│      └─ Configuration: OIDC Issuer URL & JWKS URI

Clients (federated-jwt):
├─ web-k8s-client (Authenticates using Kubernetes SA token)
└─ web-secret-client (Traditional client-secret)
```

Identity Provider Configuration

Preview Feature

```
name: KC_FEATURES
value:"client-auth-federated,kubernetes-service-accounts"
```

Establishing Trust with Kubernetes

01. IdP Creation: Create a new `kubernetes` Identity Provider instance.

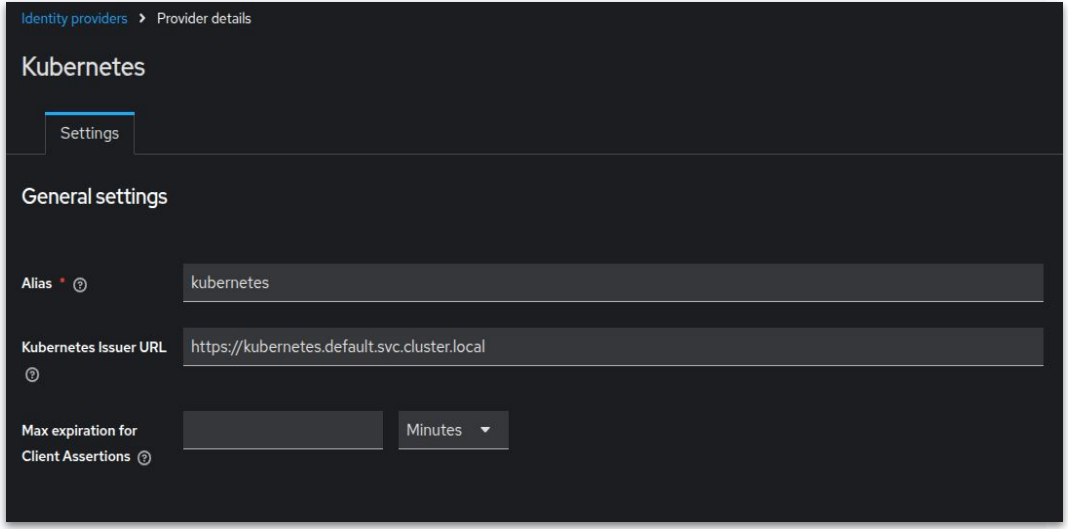
02. Key Fetching: Keycloak automatically fetches the JWKS (JSON Web Key Set) from the configured issuer URL.

```
# Create Kubernetes Identity Provider
kcadm.sh create identity-provider/instances -r kubernetes \
-s alias=kubernetes \
-s providerId=kubernetes \
-s config.issuer=https://kubernetes.default.svc.cluster.local
```

03. Claim Validation: Validates signature + claims (iss, sub, aud, exp) and maps to client using issuer + subject matching.

Establishing Trust with Kubernetes

This establishes the trust relationship from **Keycloak** to the **Kubernetes** cluster.

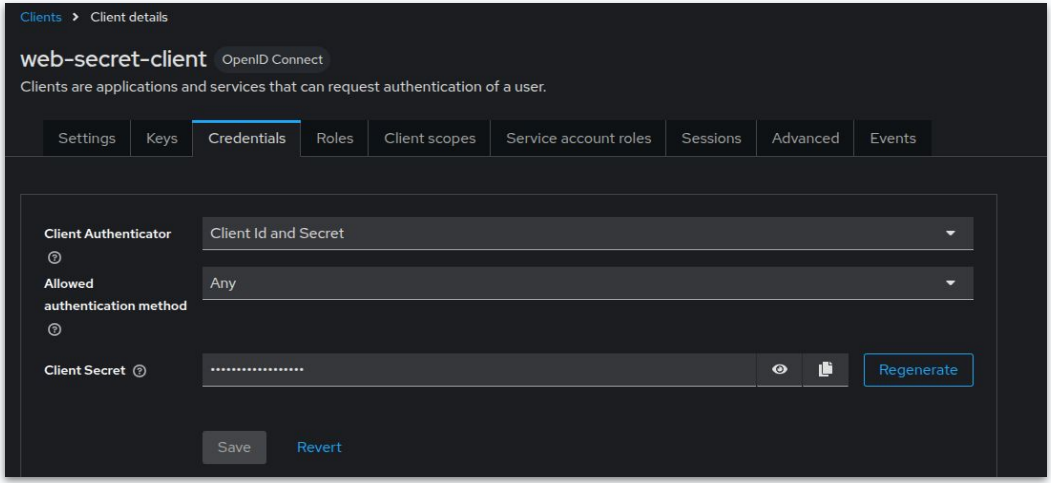


Client Configuration

Keycloak Client Configuration

BEFORE

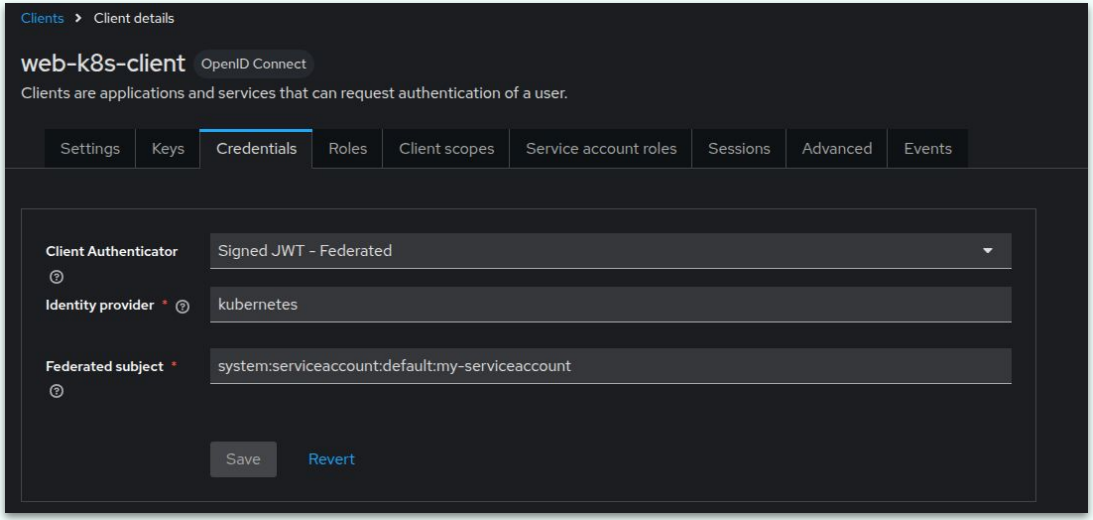
Traditional Client Secret



Keycloak Client Configuration

AFTER

Secretless Configuration



Application Configuration

Kubernetes Workload Identity using Service Account

Identity Provisioning (Kubernetes)

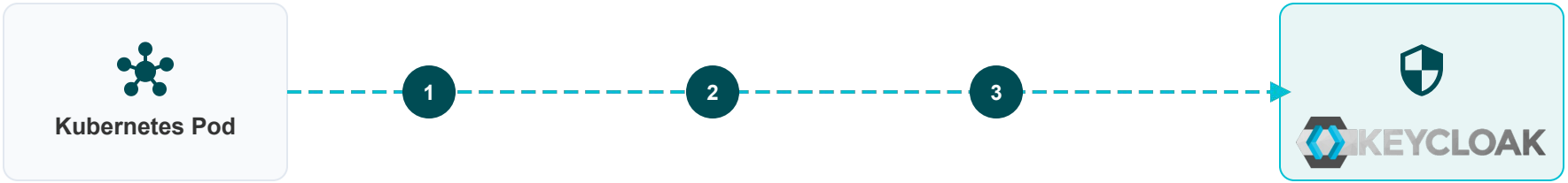
- **Projected Token:** Workload mounts a temporary Service Account token.
- **Audience Restriction:** The token's *audience* is cryptographically scoped.
- **Ephemeral Life:** Token automatically expires after 3600 seconds.

```
serviceAccountName: demo-app
volumes:
- name: keycloak-token
   projected:
     sources:
     - serviceAccountToken:
         path: token
         audience: https://keycloak/realms/k8s
         expirationSeconds: 3600
```

Keycloak Validation

- **Signature Check:** Verifies the JWT signature against the Kubernetes JWKS endpoint.
- **Audience Match:** Checks that the token's *aud* claim matches the Keycloak realm issuer URL.
- **Client Mapping:** Maps the token's *sub* (Service Account ID) to the provisioned Keycloak client.
- **Outcome:** If all checks pass, the client is authenticated, and Keycloak issues the Access Token.

Kubernetes Service Account JWT for client authentication



Step 1: Identity Injection

Workload mounts a Projected Service Account Token (short-lived, auto-rotated).

Step 2: Audience Binding

Token's *aud* claim is set specifically to the Keycloak Realm Issuer URL.

Step 3: Client Assertion

Client sends the JWT token to Keycloak as the *client_assertion* parameter.

Step 4: Trust Verification

Keycloak validates the token via configured K8s IdP (signature and subject).

Demo

The demo code is available at https://github.com/rishabhsvats/kubecon-2026
Reference Documentation: https://www.keycloak.org/2026/01/federated-client-authentication

Summary: The Core Value

The Problem

Static client secrets require manual distribution, rotation.

Solution

Federated client authentication eliminates static secrets by trusting platform-issued workload identity tokens like Kubernetes SA token or SPIFFE JWT-SVID.

Q&A

Contact Details

Name
Rishabh Singh

Position
Principal Technical Support Engineer at Red Hat

Email Address
rishabhsvats@gmail.com

Medium
medium.com/@rishabhsvats