TECHNICAL WHITE PAPER

Enterprise MFA Architecture

Keycloak, OAuth 2.0, JWT, and API-gateway enforcement for modern application platforms

Chandrakanth Anne

Review Draft • August 2026

Abstract

Multi-factor authentication succeeds only when it is treated as an end-to-end assurance system rather than an extra login screen. This paper describes an enterprise application pattern used for a Kubernetes-based billing and reconciliation platform: Keycloak centralizes authentication and factor policy, OAuth 2.0 and JSON Web Tokens carry bounded authorization context, an NGINX-class API gateway protects the external edge, and application services retain final authorization responsibility. The design separates human and workload identities, internal and external consumers, routine and high-risk actions, and normal recovery from controlled break-glass access.

> **PUBLICATION NOTE** The platform pattern and controls are based on implemented architecture work. Organization and product identifiers are intentionally omitted. Confirm the deployed factor methods, token lifetimes, and recovery procedures before external publication.

1. Why MFA Is an Architecture Concern

A password plus a second prompt does not automatically create a strong security boundary. The result depends on factor independence, enrollment, session management, token scope, gateway behavior, application authorization, reset controls, audit evidence, and the ability to survive identity-provider or network failure without creating a bypass. The architecture must also distinguish a human proving identity from a workload authenticating with a machine credential.

1.1 Design objectives

- Centralize authentication policy while keeping authorization decisions close to protected resources.
- Require MFA for privileged users and step up assurance for sensitive business actions.
- Use modern redirect-based OAuth flows with exact redirect validation and PKCE where applicable.
- Issue short-lived, audience-restricted tokens and validate them at every trust boundary.
- Separate human sessions, service accounts, internal clients, and external consumers.
- Make factor recovery, administrator break-glass access, key rotation, and audit review explicit.

2. Reference Architecture

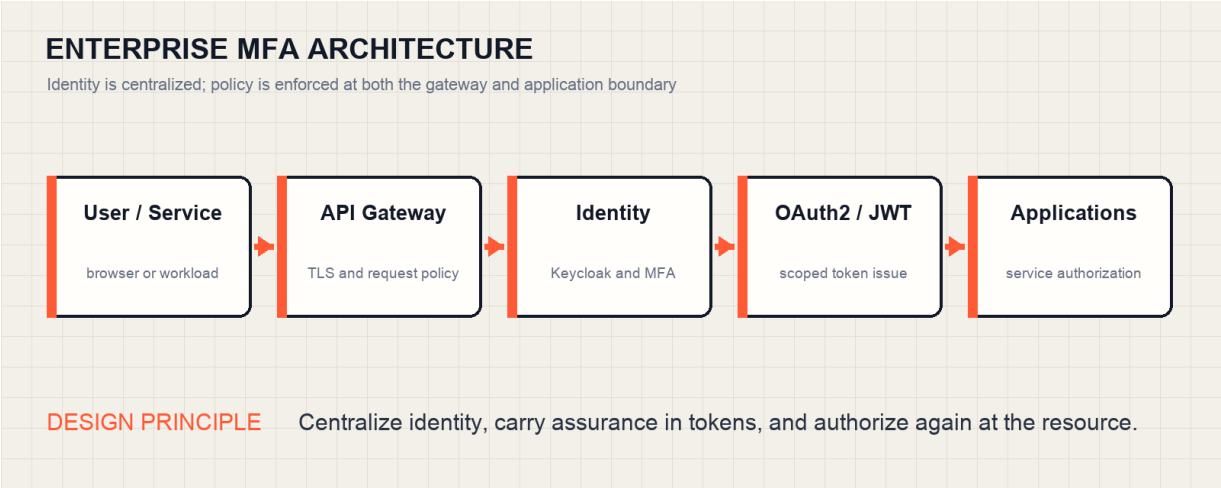

Figure 1. Authentication is centralized, but trust is never delegated to the gateway alone.

The gateway terminates or forwards TLS according to deployment policy, applies coarse client and route controls, and rejects malformed or obviously invalid requests. Keycloak owns interactive authentication, required actions, factor enrollment, step-up flows, session state, and token issuance. Protected services validate the token signature, issuer, audience, expiry, not-before time, token type, and the authorization context required for the specific operation.

2.1 Trust boundaries

Boundary	Primary responsibility	Must not assume
Client to gateway	TLS, request shape, route policy, rate controls	A presented token is valid or sufficient
Gateway to identity	Exact endpoints, protected callbacks, secure headers	A redirect is safe because it is internal
Identity to client	User authentication, MFA, consent, token issue	Every client needs the same scopes or assurance
Gateway to service	Network policy and propagated context	Gateway validation replaces service authorization
Service to data	Business authorization and tenant boundaries	Role names alone prove record-level access

3. Human and Workload Authentication

3.1 Human interactive flow

1. The client starts an authorization-code request with an exact registered redirect URI, state, nonce where OpenID Connect is used, and PKCE for public or browser-based clients.
2. Keycloak authenticates the user and evaluates the realm/client authentication flow, including the required second factor and any conditional or step-up policy.
3. The client exchanges the one-time authorization code at the token endpoint. Tokens are never returned in a URL fragment or accepted from a query string by protected APIs.
4. The gateway and service validate the resulting access token. The service maps scopes, roles, tenant, and assurance context to the requested operation.
5. Sensitive actions can require fresh authentication and a higher assurance context rather than relying on the age of the existing session.

3.2 Workload identity

Service-to-service calls do not perform human MFA. They use a distinct confidential client or workload identity, tightly scoped credentials, separate token audiences, and short lifetimes. Human roles must never be copied onto a service account merely to make an integration work. Where platform capabilities permit, key-based client authentication or workload federation is preferable to long-lived shared secrets.

4. Factor Strategy

The identity provider should own factor enrollment and verification so application services do not store OTP seeds, recovery codes, or authenticator metadata. Factor choice is policy-dependent. TOTP is widely deployable and useful as a transitional second factor, but manually entered OTP is not phishing-resistant. For privileged administration and high-impact actions, a WebAuthn or passkey-based factor provides stronger resistance by binding cryptographic authentication to the legitimate verifier.

Factor pattern	Strength	Architectural use
TOTP authenticator	Independent possession factor; susceptible to real-time phishing	Broad compatibility and transitional MFA
WebAuthn / passkey	Cryptographic and phishing-resistant when correctly configured	Preferred for administrators and high-value actions

Factor pattern	Strength	Architectural use
Client certificate	Strong device/workload identity; operationally demanding	Managed workstations, service identity, or mTLS
Recovery code	Emergency single-use fallback	Controlled recovery only; securely generated and stored

4.1 Step-up authentication

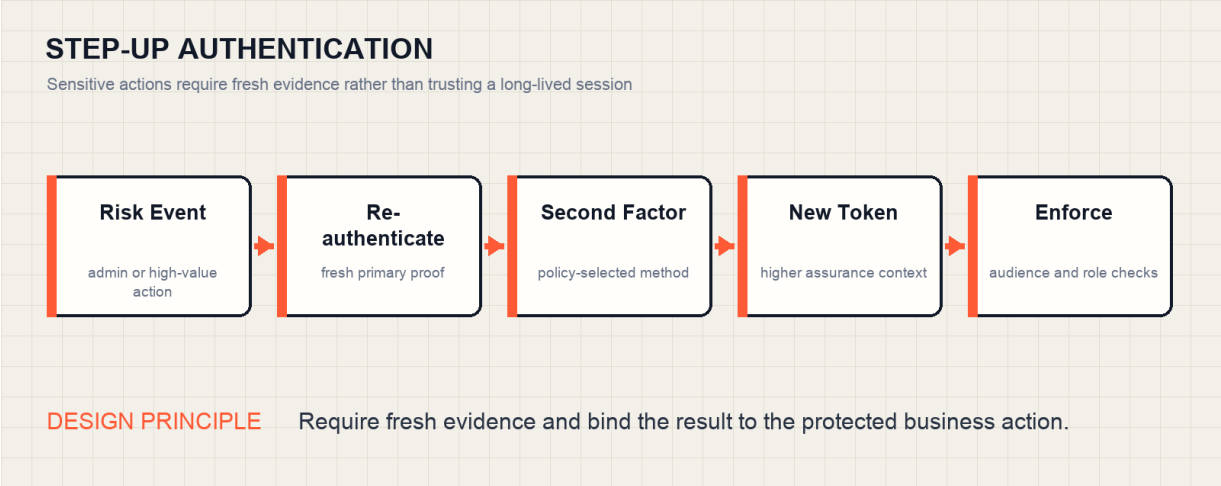

Figure 2. Step-up obtains fresh, stronger evidence before a high-risk operation is authorized.

Step-up is appropriate for administrative elevation, factor replacement, payout or billing configuration changes, data export, credential rotation, and other sensitive workflows. The client requests or is redirected into an authentication flow that produces a new token carrying an assurance context suitable for the action. The service must validate that context; merely showing a second-factor screen without binding the result to the token and operation provides weak evidence.

5. Token and Session Design

5.1 Access-token validation

- Allow only configured signing algorithms and current trusted keys.
- Verify issuer, intended audience, expiry, not-before time, and token type.
- Use scopes and roles as coarse entitlements; evaluate tenant and object-level rules in the service.
- Reject tokens from URL query strings and avoid logging authorization headers or full claims.
- Refresh signing keys safely; a key-refresh failure must not silently disable signature validation.

- Use distinct audiences and clients for internal services, external consumers, and administration.

5.2 Refresh and revocation

Access tokens should remain short-lived. Refresh tokens require rotation or sender-constraining where supported, replay detection, client binding, and revocation when the user, client, factor state, or risk posture changes. A long-lived browser session should not imply that a fresh MFA assertion exists. Services should use token assurance context and authentication time when policy requires recency.

6. Gateway and Service Enforcement

The gateway is valuable for uniform TLS policy, route admission, request size, rate limits, coarse claim checks, and protection of public identity callbacks. It is not the final policy decision point. Application services understand business operations, tenant ownership, data sensitivity, and separation-of-duties rules that a gateway cannot reliably infer.

> **DEFENSE IN DEPTH** Validate at the edge to reduce attack surface. Authorize again in the service because a misroute, internal call, or gateway configuration error must not become an authorization bypass.

6.1 Reverse-proxy controls

- Use an explicit trusted-proxy list and overwrite, rather than append to, security-sensitive forwarded headers.
- Preserve the original scheme and host only through trusted infrastructure; configure identity-provider hostname and proxy mode consistently.
- Allow exact redirect URIs. Wildcards and open redirects create token and authorization-code leakage paths.
- Apply separate routes and stronger controls to administrative endpoints; do not expose the identity administration plane through the public application gateway by default.
- Bound request bodies, connection concurrency, login attempts, and token-endpoint abuse without turning lockout into a denial-of-service tool.

7. Enrollment, Recovery, and Break-Glass

Enrollment and recovery are authentication flows and must be protected at the same level as login. Factor replacement should require an existing enrolled factor or an independently verified recovery process, notify the user out of band, invalidate relevant sessions when appropriate, and produce a high-

signal audit event. Help-desk verification must not reduce a phishing-resistant account to knowledge-based questions.

7.1 Controlled break-glass access

- Keep emergency credentials offline and separate from normal administrator identities.
- Require independent approval and record a reason, owner, activation time, and expected expiry.
- Make activation short-lived, rate-limited, tightly scoped, and visible to security monitoring.
- Rotate or destroy activated credentials after use and review every action taken during the window.
- Test the process periodically; an untested break-glass path is either unavailable or an unrecognized bypass.

8. Kubernetes and Platform Operations

In a Kubernetes deployment, the identity service, gateway, and protected applications need independent health signals, disruption budgets, secret rotation, network policies, and capacity plans. TLS keys, client secrets, and administrative credentials belong in a managed secret store and should be mounted or injected with least privilege. The platform must account for identity-provider unavailability: new interactive logins may fail, while already issued short-lived tokens can remain valid only according to explicit risk and availability policy.

Operational concern	Control	Evidence
Key rotation	Overlapping trusted keys; bounded cache refresh	Old/new key validation and rollback tests
Identity outage	Defined token-validation and login behavior	Failure-mode exercise and runbook
Secret exposure	External secret manager; least privilege	Rotation record and access audit
Gateway drift	Policy as code and peer review	Versioned config and integration tests
Session abuse	Short lifetimes, revocation, anomaly signals	Revocation and replay tests

9. Verification Strategy

A security architecture is not complete until negative behavior is tested. Verification should run against real gateway, identity, and service processes—not mocks that cannot reproduce proxy headers, key rotation, cookie policy, token audiences, or authentication-flow configuration.

Test family	Representative checks	Expected result
Authentication	Missing factor, wrong OTP, unregistered authenticator, expired challenge	No token; bounded attempts; safe event
OAuth flow	Redirect mismatch, state/nonce failure, PKCE downgrade, code replay	Flow rejected; no credential leakage
JWT validation	Wrong issuer/audience, expired/early token, disallowed algorithm	401 with generic response
Authorization	Missing role, wrong tenant, stale privilege, direct internal route	403; no protected data
Recovery	Factor replacement, recovery-code replay, help-desk bypass attempt	Independent verification and auditable denial
Operations	Key rollover, identity outage, gateway restart, clock skew	Documented bounded behavior

10. Architecture Lessons

- MFA assurance must travel with the session or token context and be evaluated where the sensitive action occurs.
- Central identity reduces duplicated credential logic, but it increases the importance of realm configuration, backup, change control, and availability engineering.
- The gateway creates a consistent edge; services remain the authority for business and tenant authorization.
- Recovery is often the weakest authentication path. Design it before rollout, not after the first locked-out administrator.
- Human and workload identities need different flows, credentials, scopes, and operational ownership.
- Phishing-resistant factors should be the target state for privileged access; OTP can be a practical transition rather than the end state.

11. Pre-Publication Review Checklist

Before publishing this implementation paper, validate the following deployment-specific details against the final architecture and security runbooks:

- The factor methods actually enabled for standard users, privileged users, and recovery.
- Authorization-code, PKCE, redirect-URI, and client-type configuration for every interactive client.
- Access-token and refresh-token lifetimes, rotation behavior, audience mapping, and assurance claims.
- The exact NGINX gateway validation responsibilities and the checks repeated inside services.
- Role, scope, tenant, and internal/external consumer mappings used by the billing platform.
- Break-glass ownership, approval, expiry, monitoring, and credential-rotation process.
- Key rotation, identity outage, session revocation, and factor-recovery test evidence.

12. Conclusion

The strongest enterprise MFA design does not ask every application to become an identity system. It centralizes factor policy and session management in a capable identity provider, applies modern OAuth security practices, constrains tokens by audience and privilege, and preserves resource-level authorization in the services that understand the business action. With disciplined recovery, step-up, gateway configuration, key management, observability, and negative testing, MFA becomes an architectural assurance layer rather than a fragile login feature.

References

1. Keycloak Server Administration Guide. https://www.keycloak.org/docs/latest/server_admin/

2. NIST SP 800-63B: Authentication and Authenticator Management. https://pages.nist.gov/800-63-4/sp800-63b.html

3. RFC 9700: Best Current Practice for OAuth 2.0 Security. https://www.rfc-editor.org/rfc/rfc9700.html

4. RFC 7636: Proof Key for Code Exchange. https://www.rfc-editor.org/rfc/rfc7636.html

5. OWASP Multifactor Authentication Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html

6. OpenID Connect Core 1.0. https://openid.net/specs/openid-connect-core-1_0.html