Migrating from Another IAM Provider to Keycloak

Migrating to Keycloak from other IAM platforms such as Auth0, Okta, Firebase Auth, or custom-built identity solutions requires careful preparation, structured data transformation, and secure reconfiguration of users, applications, and federation protocols. This guide provides a comprehensive, command-supported migration pathway tailored for real-world deployments—especially useful in DevOps pipelines and managed hosting environments such as Elestio.

Pre-Migration Preparation

Begin by auditing your existing IAM system to determine the number of users, the complexity of roles and permissions, the use of federated identity providers (like Google or LDAP), and any custom claims or attributes associated with each user. Export the data structure if the platform supports it. For example, Auth0 offers a Management API to export users in JSON format, while Okta allows CSV exports directly from the dashboard. Firebase Auth provides CLI-based user export via the auth:export command.

Simultaneously, deploy a new Keycloak instance on your preferred infrastructure—using Docker, Kubernetes, or a managed solution. For local Docker-based testing, the following command spins up a Keycloak container:

```
docker run -d --name keycloak \
  -p 8080:8080 \
  -e KEYCLOAK_ADMIN=admin \
  -e KEYCLOAK_ADMIN_PASSWORD=admin \
  quay.io/keycloak/keycloak:24.0.3 \
  start-dev
```

After starting the Keycloak server, access the admin console at http://localhost:8080/admin/. Create a new realm to isolate your identity configuration. In this realm, define the clients (applications), roles, and groups you plan to import or recreate based on your previous IAM structure.

User and Credential Migration

Export users from your existing IAM provider and structure the data for compatibility with Keycloak. If using Auth0, the export may look like this:

```
{
  "email": "jane.doe@example.com",
  "user_id": "auth0|abc123",
  "email_verified": true,
  "given_name": "Jane",
  "family_name": "Doe",
  "custom_roles": ["admin", "viewer"]
}
```

Transform this data into a Keycloak-compatible JSON using a script. You can use the Keycloak Admin REST API or the kcadm.sh CLI to programmatically create users. Here’s an example using kcadm.sh:

```
./kcadm.sh config credentials --server http://localhost:8080 \
  --realm master \
  --user admin \
  --password admin

./kcadm.sh create users -r myrealm -s username=jane.doe \
  -s enabled=true \
  -s email=jane.doe@example.com \
  -s emailVerified=true

./kcadm.sh set-password -r myrealm --username jane.doe --new-password newPassword123!
```

To bulk import users, generate a JSON file with user definitions and mount it into the Keycloak container using the keycloak-config-cli. For example:

```
docker run --rm \
  -e KEYCLOAK_URL=http://localhost:8080 \
  -e KEYCLOAK_USER=admin \
  -e KEYCLOAK_PASSWORD=admin \
  -v "$(pwd)/realm-config:/config" \
  adorsys/keycloak-config-cli:latest
```

If your previous IAM provider did not expose hashed passwords or used incompatible hashing algorithms, plan to send password reset links after user import. Alternatively, you can enforce first-login password resets using the following command:

```
./kcadm.sh update users/<user_id> -r myrealm -s "requiredActions=['UPDATE_PASSWORD']"
```

Application and Federation Migration

Next, migrate application integrations. In Keycloak, applications are known as **clients**. For each application that used your old IAM system, recreate a corresponding client in Keycloak. Choose the correct protocol (OpenID Connect or SAML) and configure the redirect URIs, web origins, client secrets, and access token lifetimes.

For example, to create a public OpenID Connect client:

```
./kcadm.sh create clients -r myrealm \
  -s clientId=my-app \
  -s enabled=true \
  -s publicClient=true \
  -s 'redirectUris=["https://myapp.com/*"]'
```

For third-party identity federation, use the Keycloak admin console or CLI to add identity providers. To connect Google OAuth:

```
./kcadm.sh create identity-provider/instances -r myrealm \
  -s alias=google \
  -s providerId=google \
  -s enabled=true \
  -s storeToken=true \
  -s "config.clientId=<GOOGLE_CLIENT_ID>" \
  -s "config.clientSecret=<GOOGLE_CLIENT_SECRET>" \
  -s "config.defaultScope=email profile"
```

For LDAP integration:

```
./kcadm.sh create user-storage -r myrealm \
  -s name=ldap \
  -s providerId=ldap \
  -s "config.connectionUrl=ldap://ldap.example.com" \
  -s "config.bindDn=cn=admin,dc=example,dc=com" \
  -s "config.bindCredential=adminpass" \
  -s "config.usersDn=ou=users,dc=example,dc=com"
```

For SAML-based federation, download the SAML metadata from your IdP and import it using the admin console under **Identity Providers > Add provider > SAML v2.0**.

Post-Migration Validation and Optimization

After users, clients, and federation setups are migrated, conduct the following checklist for validation:

- **User Login Testing**: Log in with a subset of migrated user accounts to verify that usernames, emails, roles, and group mappings are correctly preserved.
- **Token Verification**: Use JWT decoder tools to inspect access and ID tokens issued by Keycloak. Ensure claims match what applications expect.
- **Application Login Flow**: Test login, logout, and token refresh operations in all integrated applications.
- **Admin Console Review**: Confirm that users, groups, roles, and clients appear as expected in the Keycloak admin console.
- **MFA Setup**: Enable and test two-factor authentication (TOTP or WebAuthn) for relevant user roles.
- **Email Configuration**: Configure SMTP settings under **Realm Settings > Email** and verify email-based actions such as password resets or verification emails.
- **Backup Enablement**: Configure regular database backups using cron jobs, Kubernetes volumes, or your platform’s snapshot features.
- **HTTPS Enforcement**: Ensure your instance is served over TLS with valid certificates. Update keycloak.conf or reverse proxy settings accordingly.
- **Audit Logs**: Enable event logging under **Events > Settings** to monitor authentication events and system-level changes.
- **Token Lifespan Configuration**: Adjust accessTokenLifespan, refreshTokenMaxReuse, and session timeouts to fit your application needs.
- **Security Review**: Rotate all client secrets, disable default admin accounts in production, and set up firewalls to restrict admin endpoint access.

Revision #1
Created 2025-06-16 06:26:30 UTC by kaiwalya
Updated 2025-06-16 06:54:26 UTC by kaiwalya