Realm & Configuration Migration

Exporting and Importing Realms

Elestio enables seamless migration of Keycloak realms by supporting realm exports and imports. This capability is vital for backing up configurations, replicating environments, or transitioning between staging and production systems. The process ensures consistency across deployments while preserving all realm-level resources such as users, roles, groups, clients, and identity providers.

Key Steps for Exporting and Importing

Pre-Migration Preparation

Before initiating realm export or import, it’s essential to prepare both the source and target environments to ensure compatibility and prevent data loss:

- **Create an Elestio Account and Deploy Keycloak**
  Sign up at elest.io and deploy a Keycloak instance. Ensure the Keycloak version in the target environment matches the source to avoid compatibility issues during import.
- **Backup Existing Configuration**
  Always create a snapshot or export of the existing realm configuration before starting. This ensures a rollback path in case of issues during import.
- **Verify Resource Limits**
  Confirm the Elestio service has adequate CPU, RAM, and storage to accommodate the imported realm data, especially when dealing with large user bases or multiple clients.

Exporting a Realm

Keycloak provides CLI-based tools and startup parameters to export realm configurations. Elestio supports these via custom startup commands.

- **Export Using kcadm.sh (CLI)**

```
/opt/keycloak/bin/kcadm.sh config credentials --server http://localhost:8080 --realm master --user admin --password <your-password>
/opt/keycloak/bin/kcadm.sh get realms/<realm-name> > myrealm-export.json
```

This method exports the realm configuration to a JSON file.

- **Export Using Environment Variable Method (Preferred on Elestio):** You can configure the container to perform a full export on startup:

```
KEYCLOAK_IMPORT=/opt/keycloak/data/import/myrealm-export.json
```

And use the following command:

```
/opt/keycloak/bin/kc.sh export --dir /opt/keycloak/data/import --realm <realm-name> --users realm_file
```

This will export the full realm configuration including users, clients, and roles into the myrealm-export.json file.

- **Download the Export File**

After the export completes, use the Elestio dashboard or scp/rsync to download the exported JSON file from the container.

Importing a Realm into Elestio-Hosted Keycloak

Once the realm has been exported and downloaded, follow these steps to import it into your Elestio-hosted Keycloak instance:

- **Upload Exported JSON File:** Place the exported file in a volume accessible to the Elestio container (e.g., under `/opt/keycloak/data/import/)`.
- **Configure Import Environment Variable:** In the Elestio dashboard, go to your Keycloak service → **Settings → Environment Variables**, and add:

```
KEYCLOAK_IMPORT=/opt/keycloak/data/import/myrealm-export.json
```

- **Trigger Import at Startup:** Elestio will automatically import the realm during the next container restart. To do this:
  - Click **Restart Service** from the Elestio dashboard.
  - Monitor logs in real-time to ensure the import process completes successfully.

Post-Import Validation and Optimization

After importing the realm into your Elestio-hosted Keycloak instance, perform the following steps

- **Validate Realm Components:** Confirm all users, roles, groups, clients, and identity providers have been imported. Use the Keycloak Admin UI or kcadm.sh CLI to inspect the imported realm.
- **Test Application Authentication Flows:** Update client application configurations if needed. Confirm login, token exchange, and logout flows work as expected using the new realm setup.
- **Review Access Tokens and Certificates:** Ensure keys and token lifespans are properly configured. Replace any expired or incompatible certificates.
- **Enable Monitoring and Backup:** Use Elestio's built-in monitoring tools to observe performance and usage. Schedule regular backups from the dashboard to ensure data protection.
- **Apply Security Best Practices:** Rotate admin credentials. Set up IP whitelisting and firewalls via Elestio. Review and assign minimal privileges to users and service accounts.

Benefits of Using Elestio for Realm Management

- **Simplified Automation:** Elestio automates backup, monitoring, and scaling, removing manual overhead from managing Keycloak instances.
- **Secure by Default:** Instances are provisioned with firewalls, encryption, and unique passwords. Elestio keeps Keycloak up to date with critical security patches.
- **Scalable and Portable:** Realms can be exported and imported across environments with ease, enabling multi-region replication, staging-to-prod transitions, and more.
- **Performance Optimized:** Instances are pre-tuned for performance. Elestio supports scaling CPU, RAM, and volume size based on identity workload.

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.

Cloning a Realm to a New Cluster or Region

In scenarios where high availability, regional redundancy, or environment separation (e.g., staging to production) is required, cloning an entire Keycloak realm to a new cluster or region becomes essential. This process involves exporting the realm’s configuration and optionally user data, transferring it securely, and importing it into a fresh Keycloak instance. This guide covers all the required steps, including command-line tooling, configuration handling, and validation checks to ensure a seamless realm replication.

Pre-Cloning Preparation

Before initiating the cloning process, ensure that both the source and target Keycloak clusters are accessible and running compatible versions of Keycloak. This avoids schema mismatches and import errors. Install the Keycloak Admin CLI (kcadm.sh) and Keycloak Configuration CLI (keycloak-config-cli) on your local system or CI/CD pipeline.

Deploy a new Keycloak instance in the target cluster or region. This can be done using Docker, Kubernetes, or a managed hosting provider. Example Docker command to spin up a development instance:

```
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
```

Ensure that network connectivity exists between your machine and the target Keycloak instance. Also, create an admin user for the new instance.

Exporting Realm Configuration from the Source Cluster

To begin the cloning process, export the realm’s full configuration (including clients, roles, groups, and optionally users) using the Keycloak Admin CLI or built-in export tools. If using the Keycloak start command with --export flag, execute:

```
/opt/keycloak/bin/kc.sh export --dir /opt/keycloak/data/export \
  --realm myrealm \
  --users skip
```

To include users in the export:

```
/opt/keycloak/bin/kc.sh export --dir /opt/keycloak/data/export \
  --realm myrealm \
  --users all
```

If the Keycloak instance is running in Docker:

```
docker exec -it keycloak /opt/keycloak/bin/kc.sh export \
  --dir /opt/keycloak/data/export \
  --realm myrealm \
  --users all
```

The exported realm will be saved as a JSON file, e.g., myrealm-realm.json.

Transferring Exported Data

Once exported, securely transfer the generated realm export directory or file (myrealm-realm.json) to the target cluster or region. Depending on your infrastructure, use one of the following methods:

- SCP or SFTP for VM-to-VM transfer:

```
scp myrealm-realm.json user@target-host:/tmp/
```

- AWS S3, Azure Blob Storage, or GCS for multi-cloud environments.
- Git repositories or artifact registries for CI/CD pipelines.

Ensure that the target Keycloak container or pod has access to the file location.

Importing Realm into the Target Cluster

To import the realm into the new Keycloak instance, use the same kc.sh tool on the target side. The import must be triggered before the Keycloak server starts. If using a container:

```
docker run -d --name keycloak-new \
  -p 8080:8080 \
  -e KEYCLOAK_ADMIN=admin \
  -e KEYCLOAK_ADMIN_PASSWORD=admin \
  -v $(pwd)/export:/opt/keycloak/data/import \
  quay.io/keycloak/keycloak:24.0.3 \
  start-dev --import-realm
```

This command will initialize the new Keycloak instance with the cloned realm, including users if they were part of the original export.

Alternatively, if the server is already running, use kcadm.sh or the keycloak-config-cli for post-start import. The keycloak-config-cli is suitable for GitOps-style deployments:

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

Post-Cloning Validation

After the import is complete, validate the integrity of the cloned realm with the following checks:

- Confirm that the new realm appears in the admin console and is accessible at /realms/myrealm.
- Inspect clients to verify that client IDs, redirect URIs, and secrets have been preserved.
- Validate that roles, groups, and permissions are correctly replicated.
- Test login using a few sample user accounts if users were exported.
- Decode access tokens to confirm the correctness of claims, issuer, and audience.
- Check identity provider connections (e.g., Google, LDAP) and test federated logins.
- Enable auditing under **Events > Settings** to monitor realm activity in the new instance.
- Update baseUrl settings for clients if moving across DNS regions.
- Ensure SMTP settings, themes, and custom scripts are present and functioning.
- Enable TLS and update public frontend URLs if applicable.
- Verify realm-specific settings like session timeout, brute force detection, and required actions.