Integrating OAuth Proxy, Keycloak, and PostgreSQL for Secure Authentication

Rahul Ahlawat

Introduction

This document serves as a comprehensive guide for deploying an OAuth Proxy using Keycloak for identity management. The setup is backed by a PostgreSQL database and managed through HAProxy, all hosted on a private VM.

HAProxy is positioned on a public VM to facilitate authentication requests to Keycloak from physical machines. The user workflow is as follows: when users access the portal, they first reach the OAuth endpoint, are redirected to Keycloak for authentication, and then return to the portal.

This guide details the purpose of each component, outlines the desired user workflow, and discusses potential issues that may arise in a private environment.

Prerequisites

- **Podman**: Ensure Podman is installed on your server to manage containers.
- **Network Access**: Ensure that your private VM can resolve domain names and has outbound internet access for downloading images.
- **Basic Terminal Knowledge**: Familiarity with using the command line will help execute the provided scripts.

Architecture Overview

The architecture consists of several components:

- **Postgres**: A relational database that stores Keycloak data.
- **Keycloak**: An open-source identity and access management solution.
- **OAuth Proxy**: A reverse proxy that secures applications using OAuth2.
- **HAProxy**: A high-performance TCP/HTTP load balancer that routes requests to the appropriate backend services.

Desired Workflow

1. **User Access**: A user attempts to access the portal at `https://portal.uat.india.gov.in`
2. **Routing to OAuth Proxy**: The request first hits the HAProxy server.
3. **OAuth Proxy Authentication**: HAProxy forwards the request to the OAuth Proxy, which checks if the user is authenticated.
4. **Keycloak Authentication**: If the user is not authenticated, the OAuth Proxy redirects them to Keycloak for login.
5. **Token Handling**: Upon successful authentication, Keycloak issues an access token and redirects the user back to the OAuth Proxy.
6. **Access Granted**: The OAuth Proxy validates the token and allows the user to access the portal.

Possible Causes of Issues

When operating in a private VM environment, several issues may arise that can affect the setup:

1. **Network Configuration**:
   - **Firewall Rules**: Ensure that the firewall allows traffic on necessary ports (e.g., 80, 443 for HAProxy, 8081 for Keycloak).
   - **Routing Issues**: If the private VM cannot access external resources, it may affect image downloads or updates.
2. **Service Communication**:
   - **Container Networking**: Ensure that containers are in the same Podman network or can communicate with each other.
   - **DNS Resolution**: If internal DNS is not properly set up, services may fail to resolve each other's addresses.
3. **Authentication Failures**:
   - **Keycloak Misconfiguration**: Incorrect client settings in Keycloak can lead to authentication failures.
   - **Token Expiration**: Tokens may expire quickly if not configured properly.
4. **SSL/TLS Issues**:
   - **Certificate Mismatch**: Ensure SSL certificates are correctly configured and match the domain.
   - **Insecure Connections**: Ensure that SSL is enforced if required; otherwise, it may lead to security vulnerabilities.

5. **Resource Limitations**:
   - **Insufficient Resources**: If the VM does not have enough CPU or memory, it can cause slowdowns or failures in service.

Setup Details

Postgres Container

```
#!/bin/bash

if ! podman volume inspect postgres-data &>/dev/null; then
    podman volume create postgres-data
fi

podman run -dt \
    --name keycloak-postgres --pod oauth-keycloak \
    -e POSTGRES_DB=postgres \
    -e POSTGRES_USER=postgres \
    -e POSTGRES_PASSWORD={password} \
    -e PGDATA=/var/lib/postgresql/data/pgdata \
    -v postgres-data:/var/lib/postgresql/data \
    docker.io/postgres:latest
```

Explanation

- **Volume**: A persistent volume `postgres-data` is created to store Postgres data.
- **Environment Variables**: Configured to set up the database, user, and password.

Notes

- Replace `{password}` with a strong, secure password.

Keycloak Container Configuration

```bash
#!/bin/bash

podman run -dt --name keycloak --pod oauth-keycloak \
    -e KEYCLOAK_ADMIN=admin \
    -e KEYCLOAK_ADMIN_PASSWORD={password} \
    -e KC_DB=postgres \
    -e KC_DB_USERNAME=postgres \
    -e KC_DB_PASSWORD={password} \
    -e KC_DB_URL=jdbc:postgresql://{private_vm -IP}/postgres \
    -e KC_PROXY=edge \
    -e KEYCLOAK_FRONTEND_URL={Hostname} \
    -e KEYCLOAK_HOSTNAME={Hostname} \
    quay.io/keycloak/keycloak:25.0.1 \
    start-dev --proxy=edge \
              --http-enabled=true \
              --hostname-strict=false
```

Explanation

- **Admin Credentials**: Necessary for managing Keycloak.
- **Database Configuration**: Specifies how Keycloak connects to Postgres.

Notes

- Replace placeholders with actual values.

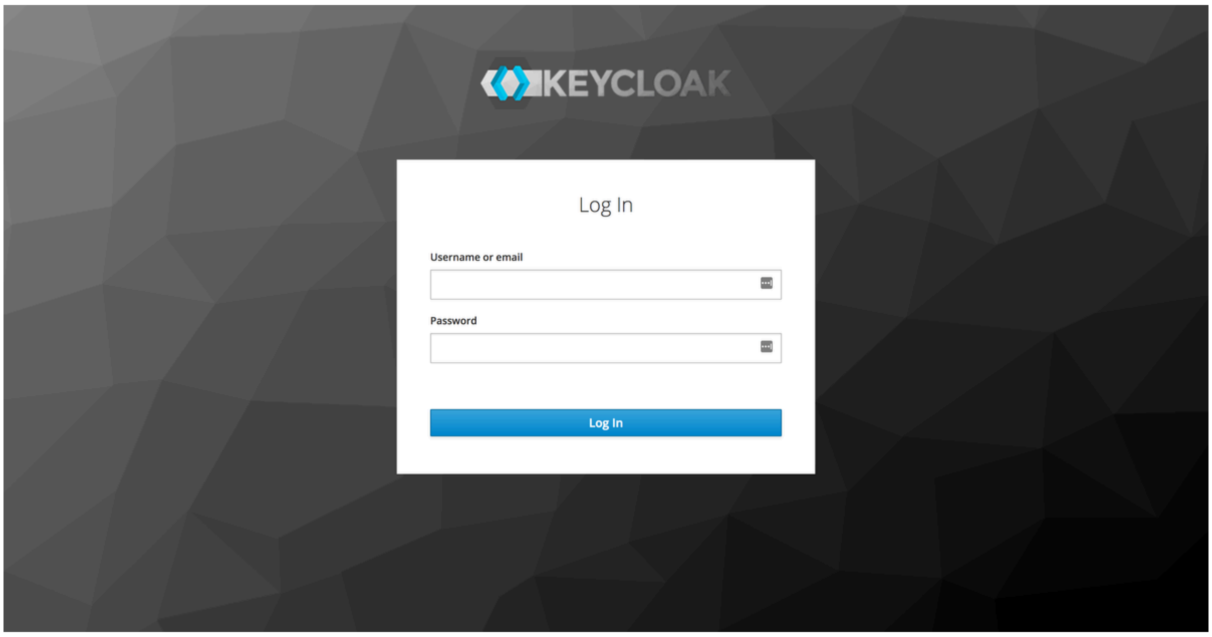
Keycloak Configuration:-

Step 1: Keycloak Login Page

Explanation: This is the login interface of Keycloak where you enter your admin credentials.

Steps:

1. Go to the URL: `http://<your-keycloak-server>:8080`
2. Enter your admin username and password.
3. Click on the "LogIn" button.

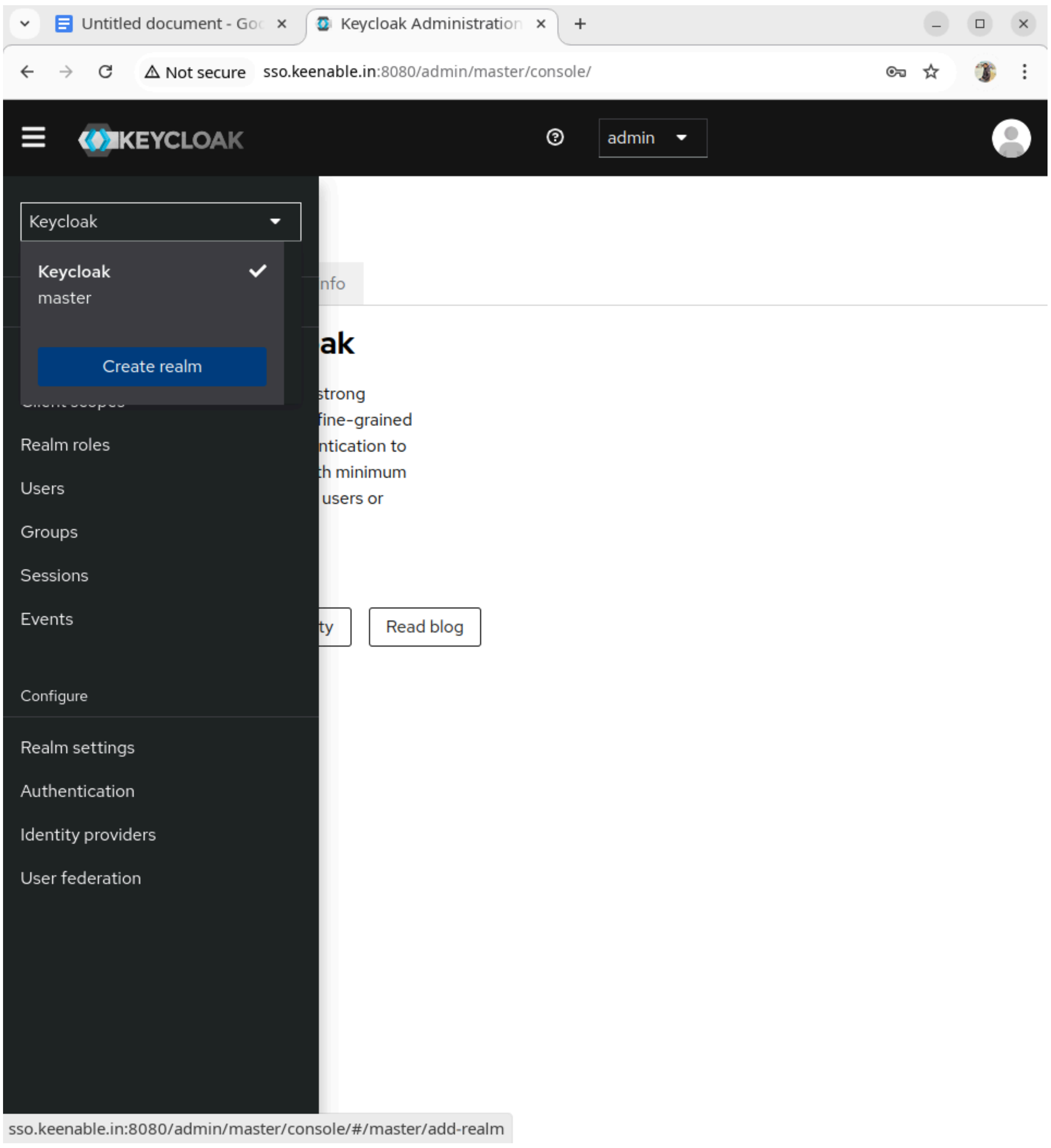


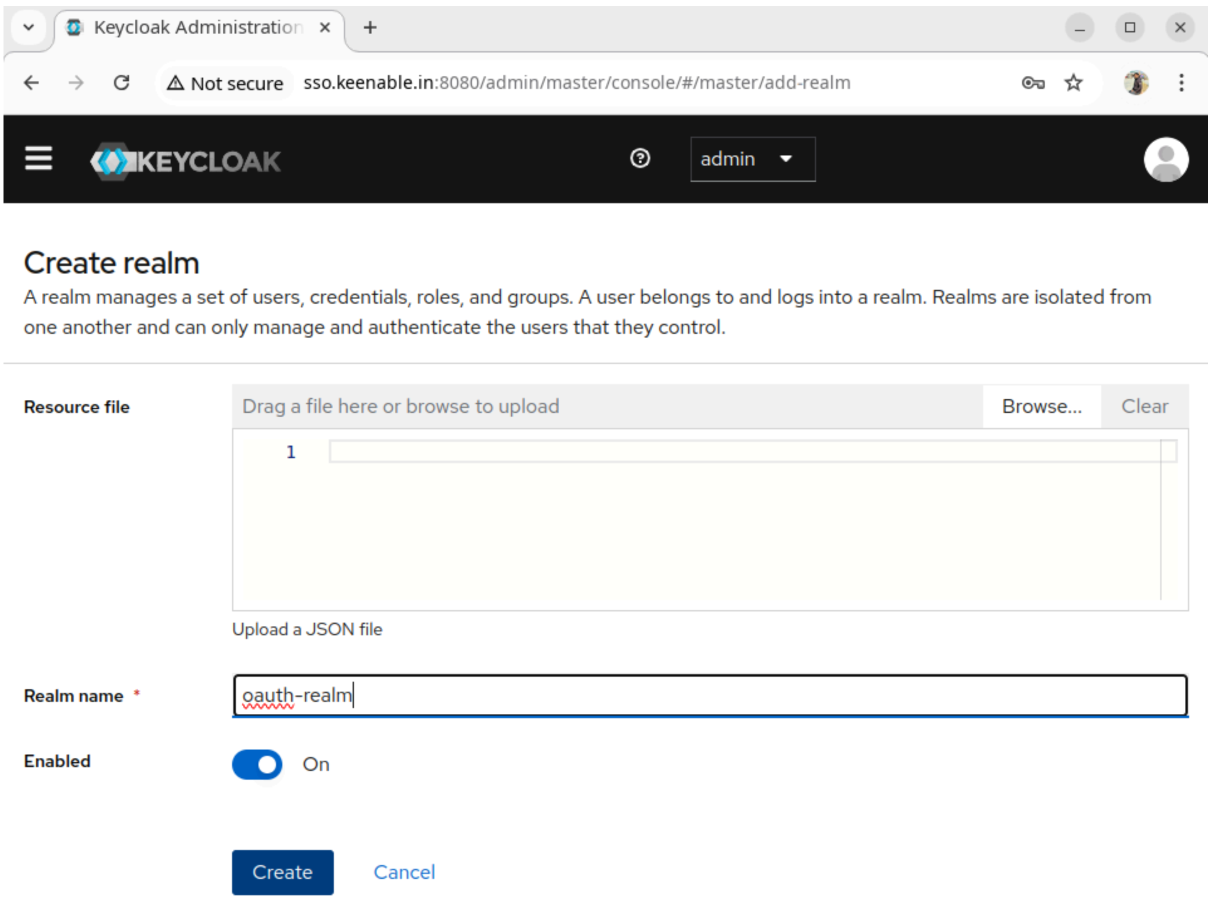
Step 2: Realm Creation

Explanation: Here, you can create a new realm, which is a logical grouping of users and applications.

Steps:

1. Click on “Create Realm."
2. Enter a name for the realm (e.g., "my-realm").
3. Click on the "Create" button.


Untitled document - Goo
Keycloak Administration
Not secure
sso.keenable.in:8080/admin/master/console/
KEYCLOAK
admin
Keycloak
Keycloak
master
Create realm
Realm roles
Users
Groups
Sessions
Events
Configure
Realm settings
Authentication
Identity providers
User federation
nfo
ak
strong
fine-grained
ntication to
h minimum
users or
ty
Read blog
sso.keenable.in:8080/admin/master/console/#/master/add-realm


Keycloak Administration

Not secure sso.keenable.in:8080/admin/master/console/#/master/add-realm

KEYCLOAK admin

Create realm

A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.

Resource file: Drag a file here or browse to upload | Browse... | Clear

Upload a JSON file

Realm name *: oauth-realm

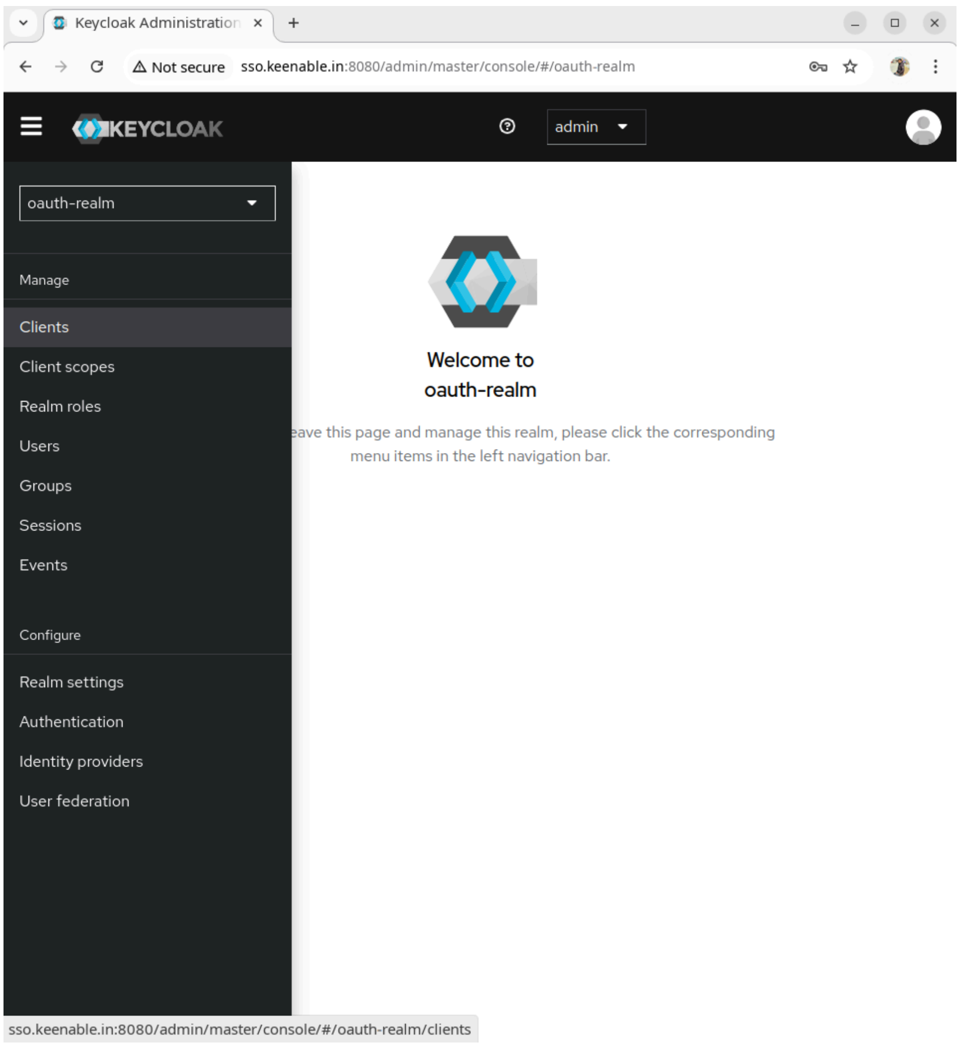
Enabled: On

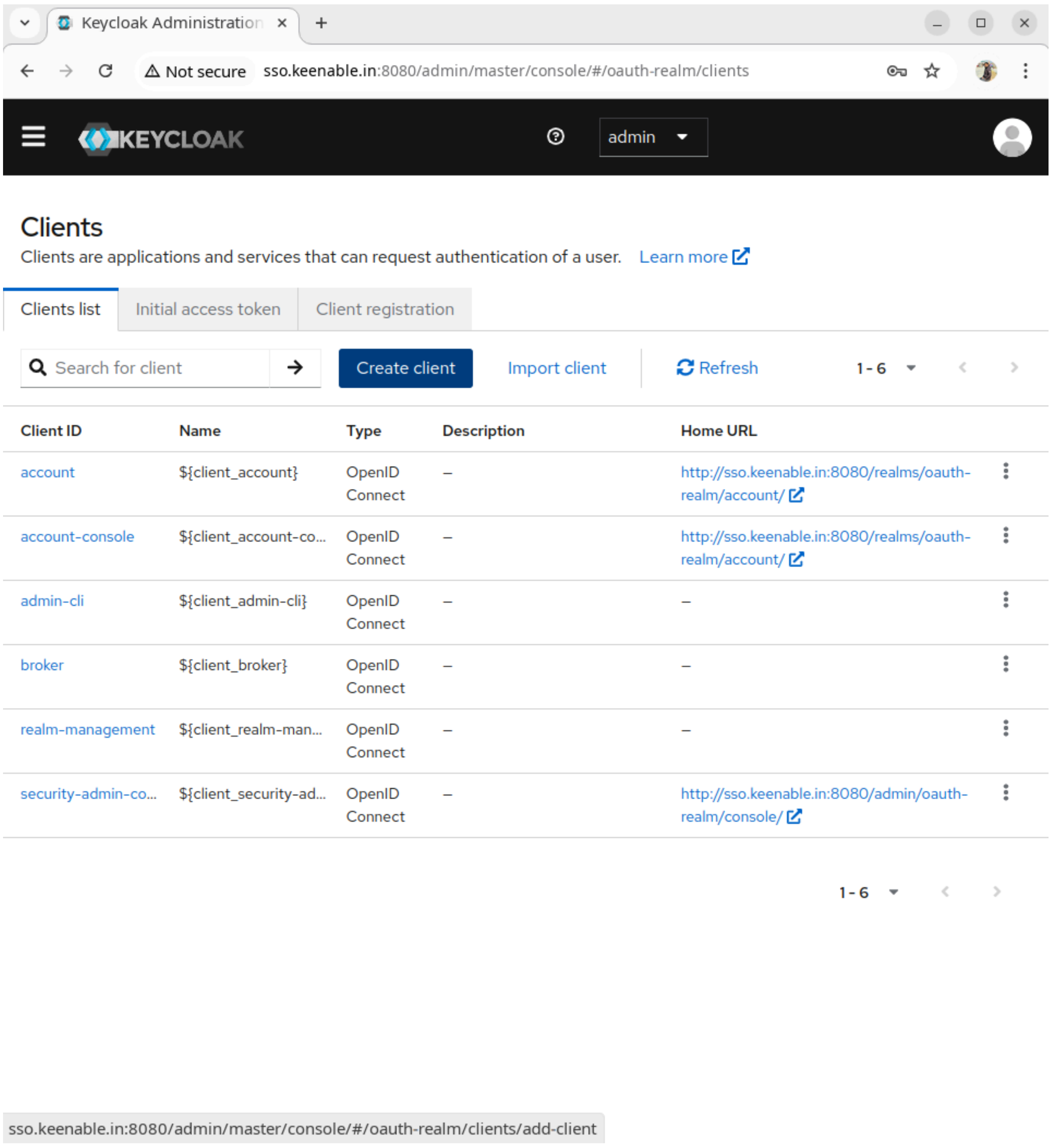
Step 3: Client Configuration

Explanation: In this section, you configure client applications that will integrate with Keycloak.

Steps:

1. Navigate to the "Clients" section on the realm dashboard.
2. Click on the "Create" button.
3. Enter a Client ID and select a protocol (e.g., "openid-connect").
4. Click on the "Save" button.


Keycloak Administration
Not secure sso.keenable.in:8080/admin/master/console/#/oauth-realm
KEYCLOAK
admin
oauth-realm
Manage
Clients
Client scopes
Realm roles
Users
Groups
Sessions
Events
Configure
Realm settings
Authentication
Identity providers
User federation
Welcome to oauth-realm
eave this page and manage this realm, please click the corresponding menu items in the left navigation bar.
sso.keenable.in:8080/admin/master/console/#/oauth-realm/clients


Keycloak Administration

Not secure sso.keenable.in:8080/admin/master/console/#/oauth-realm/clients

admin

Clients

Clients are applications and services that can request authentication of a user. Learn more

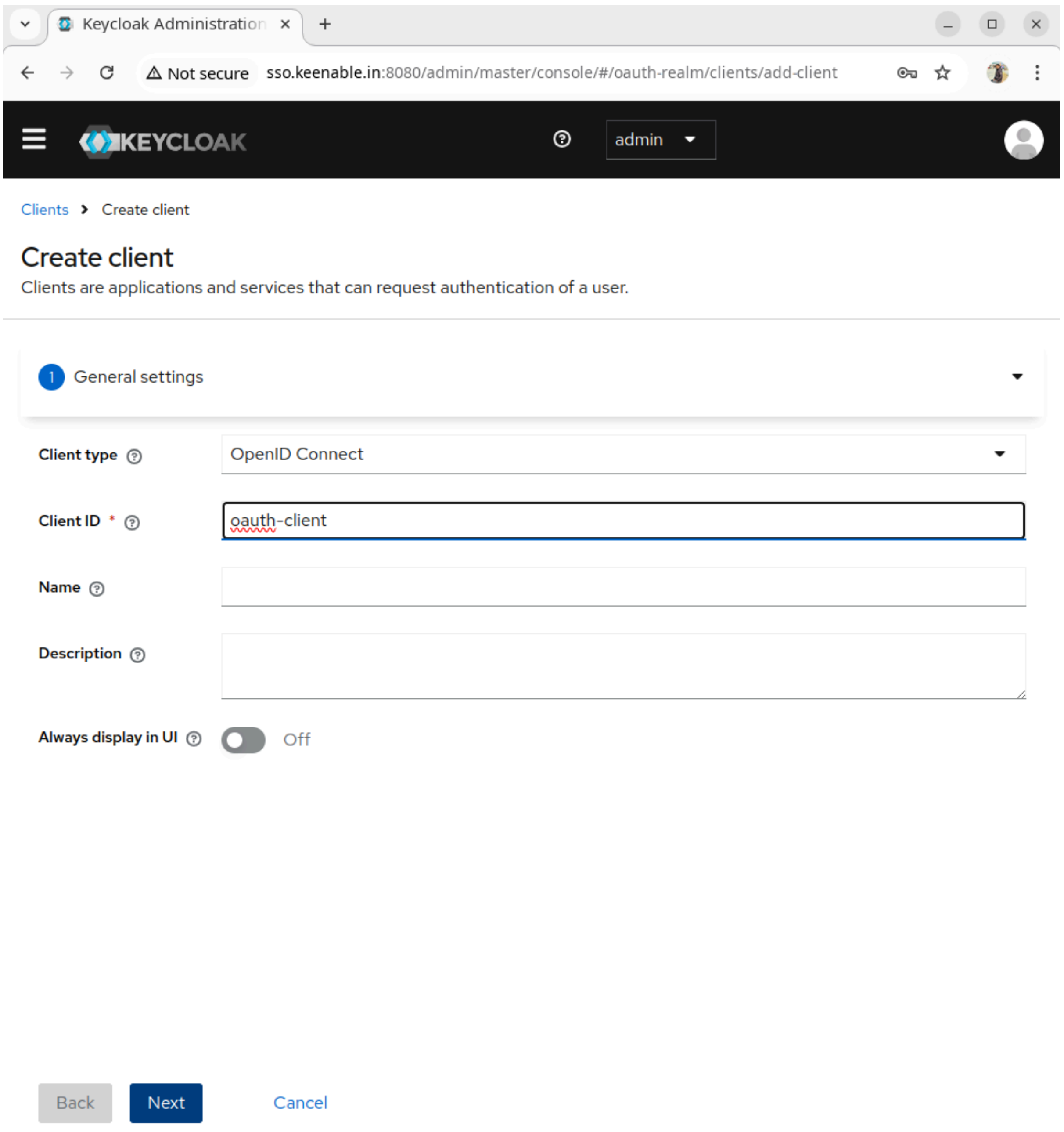
Clients list | Initial access token | Client registration

Search for client | Create client | Import client | Refresh | 1 - 6

Client ID	Name	Type	Description	Home URL
account	${client_account}	OpenID Connect	–	http://sso.keenable.in:8080/realms/oauth-realm/account/
account-console	${client_account-co...	OpenID Connect	–	http://sso.keenable.in:8080/realms/oauth-realm/account/
admin-cli	${client_admin-cli}	OpenID Connect	–	–
broker	${client_broker}	OpenID Connect	–	–
realm-management	${client_realm-man...	OpenID Connect	–	–
security-admin-co...	${client_security-ad...	OpenID Connect	–	http://sso.keenable.in:8080/admin/oauth-realm/console/

1 - 6

sso.keenable.in:8080/admin/master/console/#/oauth-realm/clients/add-client

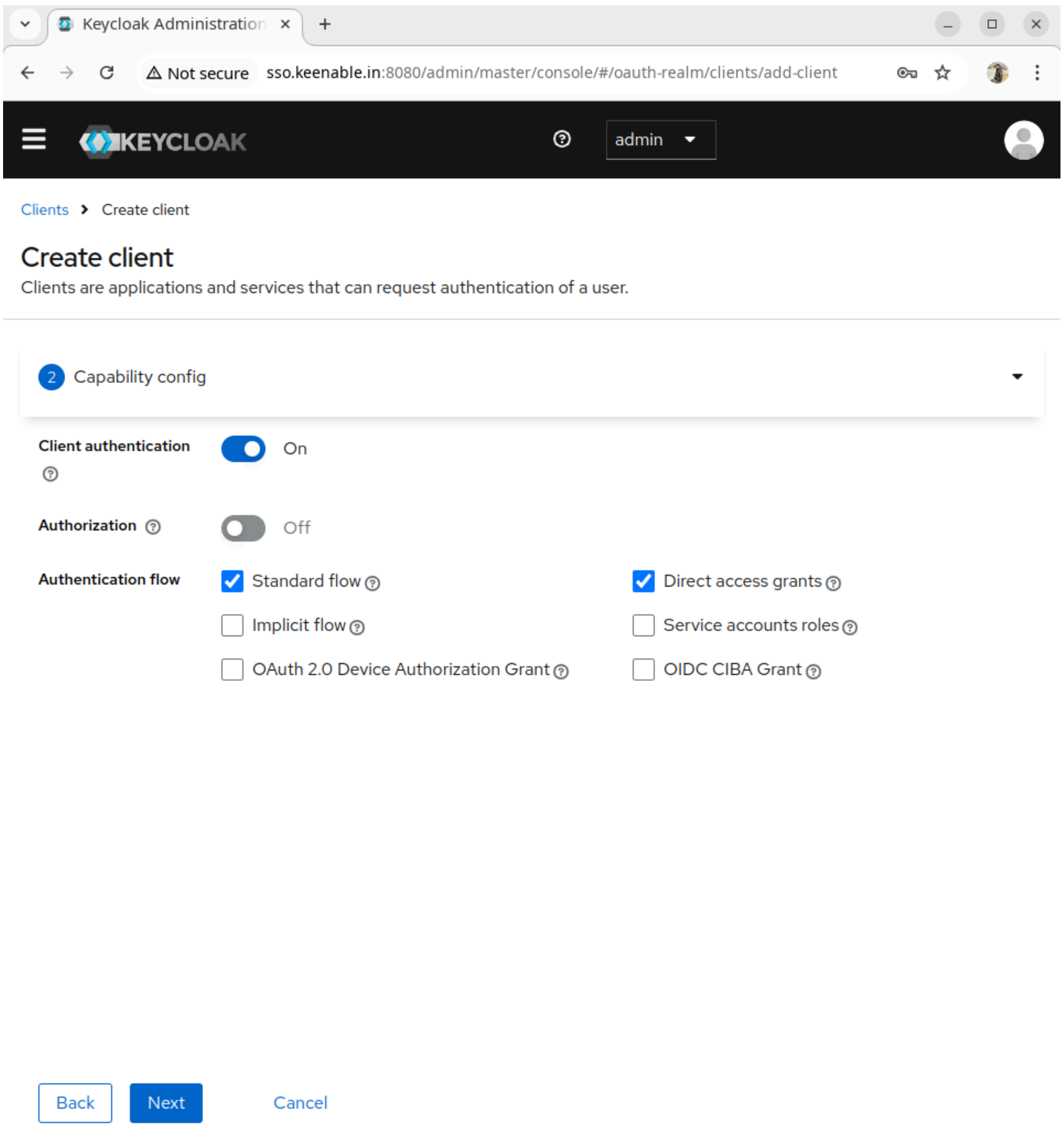

Keycloak Administration
Not secure
sso.keenable.in:8080/admin/master/console/#/oauth-realm/clients/add-client
KEYCLOAK
admin
Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

1 General settings

Client type	OpenID Connect
Client ID *	oauth-client
Name	
Description	
Always display in UI	Off


Keycloak Administration

⚠ Not secure sso.keenable.in:8080/admin/master/console/#/oauth-realm/clients/add-client

KEYCLOAK admin

Clients > Create client

Create client

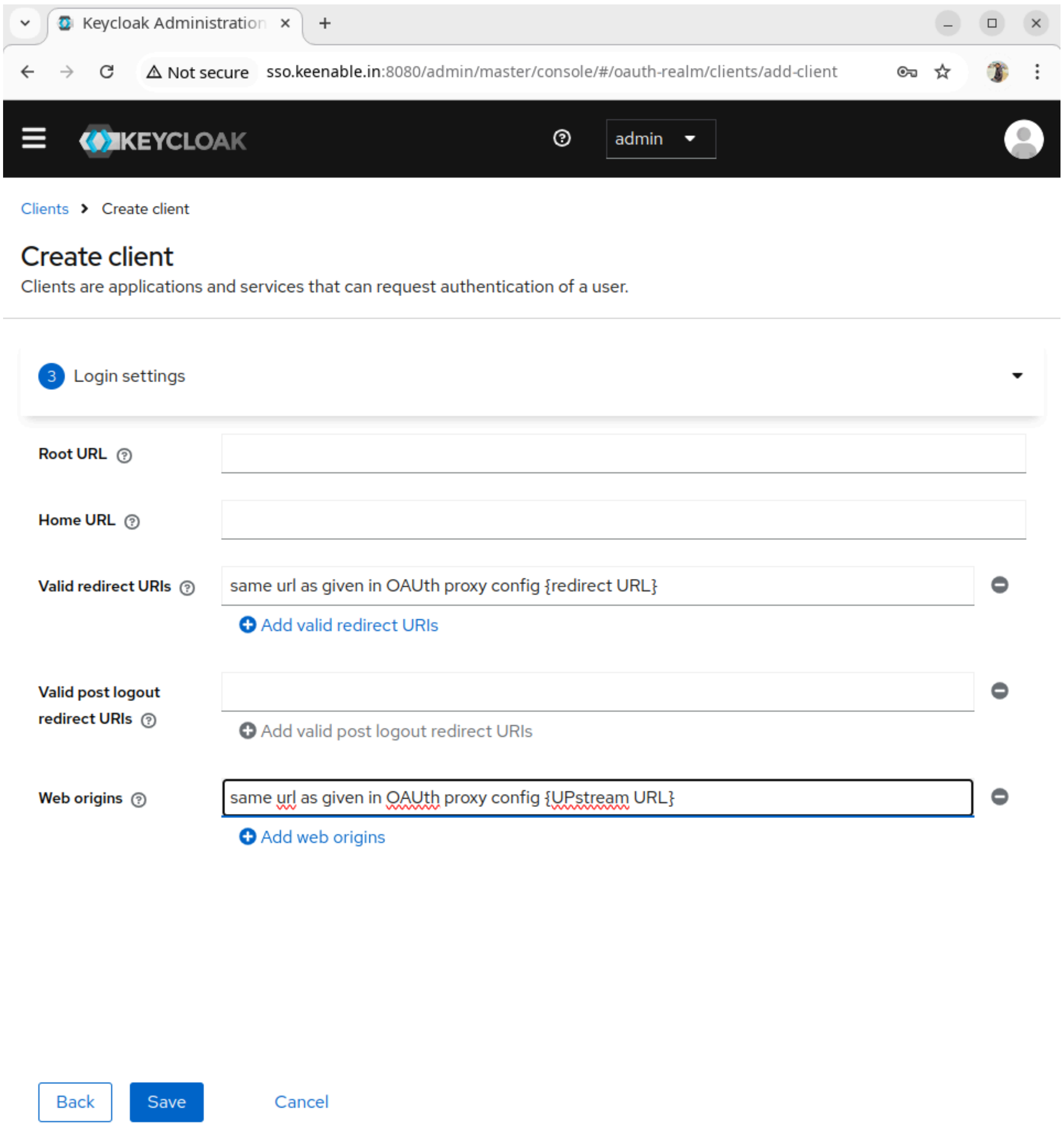
Clients are applications and services that can request authentication of a user.

2 Capability config

Client authentication: On

Authorization: Off

Authentication flow:
- [✓] Standard flow
- [✓] Direct access grants
- [ ] Implicit flow
- [ ] Service accounts roles
- [ ] OAuth 2.0 Device Authorization Grant
- [ ] OIDC CIBA Grant



Keycloak Administration

Not secure sso.keenable.in:8080/admin/master/console/#/oauth-realm/clients/add-client

KEYCLOAK admin

Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

3 Login settings

Root URL:

Home URL:

Valid redirect URIs: same url as given in OAUth proxy config {redirect URL}

Add valid redirect URIs

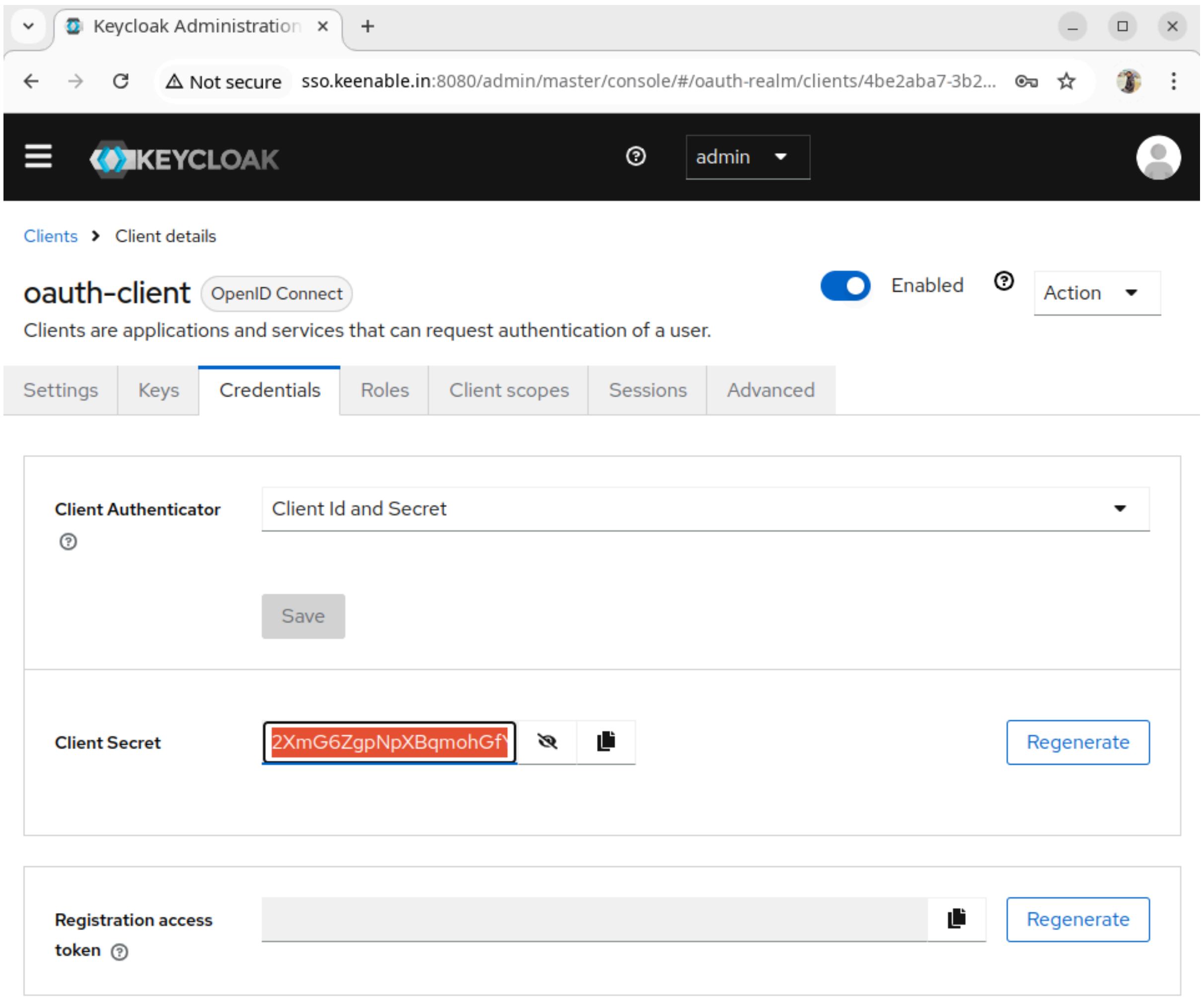
Valid post logout redirect URIs:

Add valid post logout redirect URIs

Web origins: same url as given in OAUth proxy config {UPstream URL}

Add web origins

Back Save Cancel


Keycloak Administration

Not secure sso.keenable.in:8080/admin/master/console/#/oauth-realm/clients/4be2aba7-3b2...

KEYCLOAK admin

Clients > Client details

oauth-client OpenID Connect

Enabled Action

Clients are applications and services that can request authentication of a user.

Settings | Keys | Credentials | Roles | Client scopes | Sessions | Advanced

Client Authenticator: Client Id and Secret

Save

Client Secret: 2XmG6ZgpNpXBqmohGf

Regenerate

Registration access token

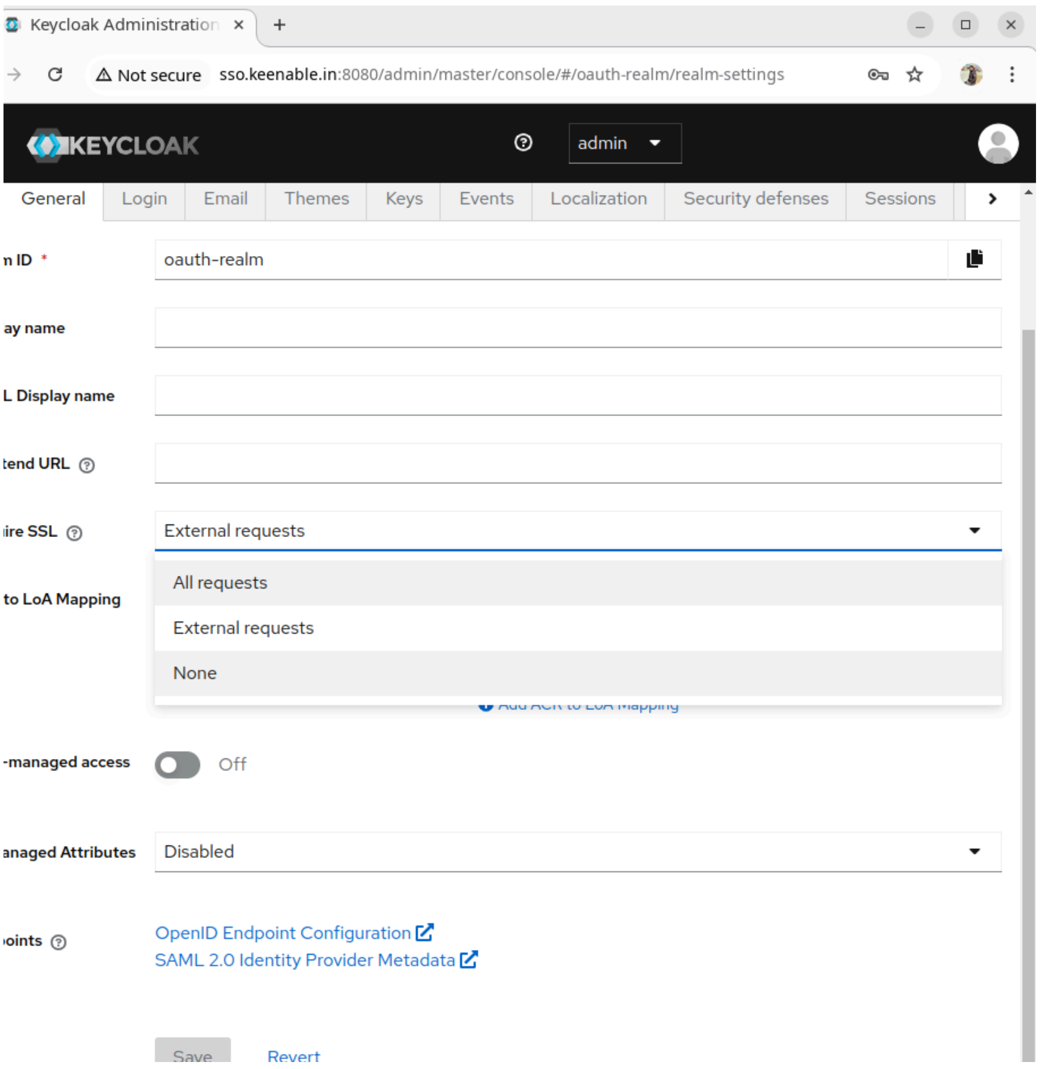
Regenerate

Step 4: SSL Configuration

Explanation: In this step, you configure Keycloak to run without SSL (HTTPS), which is useful for development environments but not recommended for production.

Steps:

1. Open the Keycloak admin console.
2. Go to the "Realm Settings" section.
3. Navigate to the "Security Defenses" tab.
4. Under "SSL Required," select the option for "None."
5. Click on "Save" to apply the changes.



n ID * oauth-realm

ay name

L Display name

tend URL

ire SSL External requests

All requests
External requests
None

to LoA Mapping

-managed access Off

anaged Attributes Disabled

oints

OAuth Proxy config file & Container

OAuthproxy.cfg File

```
## Server Address Configuration
http_address = ":4180"

## OAuth Redirect URL (OAuth2 callback)
redirect_url = "https://portal.uat.india.gov.in"

## Upstream Services (Authenticated Requests will be passed here)
upstreams = ["https://portal.uat.india.gov.in"]

## Pass User and Authentication Headers to Upstream
pass_basic_auth = true
pass_user_headers = true
pass_host_header = true
pass_access_token = true

## Allow any Email Domain for Authentication
email_domains = ["*"]

## OIDC Provider Configuration (Keycloak Settings)
provider = "keycloak-oidc"
oidc_issuer_url = "http://{public_vm -IP}:8081/realms/oauth-realm"
client_id = "oauth-client"
client_secret = "{client-secret}"
#email-domain="fosteringlinux.com"
## Secure Cookie Settings

cookie_name = "_oauth2_proxy"
cookie_secret = "{cookie-secret}"  {base64 encoded}
cookie_expire = "168h"
cookie_refresh = "1h"
cookie_secure = false
cookie_httponly = true
cookie_samesite = "lax"

## Enable CSRF Protection
```

```
disable_csrf = false

## Authorization Header
pass_authorization_header = true
set_authorization_header = true

## PKCE Code Challenge (Improves Security)
code_challenge_method = "S256"
reverse-proxy=true

## Optional: SSL Settings
ssl_insecure_skip_verify = true
```

Explanation

- **Redirect URL**: Where users are redirected after authentication.
- **OIDC Configuration**: Defines how the proxy interacts with Keycloak.

Container OAuth Proxy

```
#!/bin/bash

podman run -d --name oauth2-proxy-I \
    --pod oauth-keycloak \
    -v
/home/azureuser/oauth2-proxy/oauth2-proxy.cfg:/etc/oauth2-proxy/oauth2-p
roxy.cfg \
    quay.io/oauth2-proxy/oauth2-proxy:v5.1.1 \
    --config /etc/oauth2-proxy/oauth2-proxy.cfg
```

ADD SSL into OAuth Proxy Container

```
podman cp caserver.crt {oauth-proxy-container-name}
podman exec -it -u root {oauth-proxy-container-name} sh
```

```
cat caserver.crt  >> /etc/ssl/certs/ca-certificates.crt
```

HAproxy config File

```
global
      log /dev/log      local0
      log /dev/log      local1 notice
      chroot /var/lib/haproxy
      stats socket /run/haproxy/admin.sock mode 660 level admin
expose-fd listeners
      stats timeout 30s
      user haproxy
      group haproxy
      daemon


defaults
      log   global
      mode  http
      option      httplog
      option      dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
      errorfile 400 /etc/haproxy/errors/400.http
      errorfile 403 /etc/haproxy/errors/403.http
      errorfile 408 /etc/haproxy/errors/408.http
      errorfile 500 /etc/haproxy/errors/500.http
      errorfile 502 /etc/haproxy/errors/502.http
      errorfile 503 /etc/haproxy/errors/503.http
      errorfile 504 /etc/haproxy/errors/504.http
##image.uat.india.gov.in -->10.2.0.5:443




listen ingress
  bind *:443 ssl crt /etc/haproxy/certs/combined.pem
  mode tcp
  option tcplog
  tcp-request inspect-delay 5s
  tcp-request content capture req.ssl_sni len 25
  tcp-request content accept if { req.ssl_hello_type 1 }
  use_backend image if { hdr(host) -i image.uat.india.gov.in }
  use_backend oauth if { hdr(host) -i portal.uat.india.gov.in }


backend oauth
  server oauth2_proxy_server {private_vm -IP}:4180 check

frontend keycloak_front
    bind *:8081
    mode http
    option forwardfor
    http-request set-header X-Forwarded-Proto http
    http-request set-header X-Forwarded-Host %[req.hdr(host)]
    http-request set-header X-Forwarded-For %[src]
    default_backend keycloak_back

backend keycloak_back
    mode http
    balance roundrobin
    cookie SERVERID insert indirect nocache
    server keycloak_server {private_vm -IP}:8081 check cookie keycloak1
```

Testing the Setup

1. **Access the Portal**: Visit `https://portal.uat.india.gov.in`.
2. **Login Flow**: Ensure the user is redirected to Keycloak for authentication.
3. **Access Verification**: Confirm access to the portal after successful login.

Conclusion

This guide outlines the comprehensive setup of OAuth Proxy, Keycloak, and HAProxy on a private VM. Understanding the desired workflow and potential issues in such an environment is crucial for ensuring a smooth and secure user experience. With the right configurations and practices, you can create a robust authentication mechanism for your applications.