Member-only story

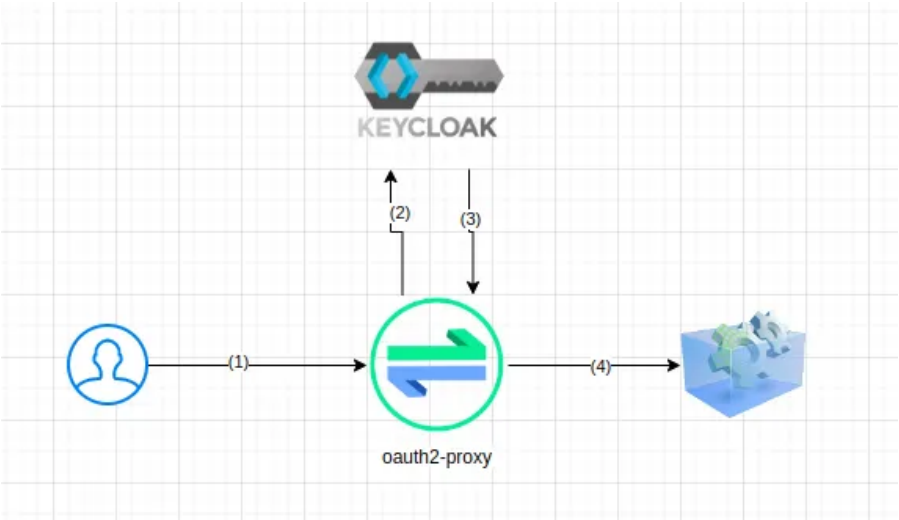
Kubernetes — Security — SSO Authentication using OAuth2 Proxy and Keycloak

Genesta Sebastien · Follow

5 min read · Apr 10, 2024

Listen Share More

This article deals with how to easily setup authentication for your applications using OAuth2 Proxy (and Keycloak as OAuth2 provider).



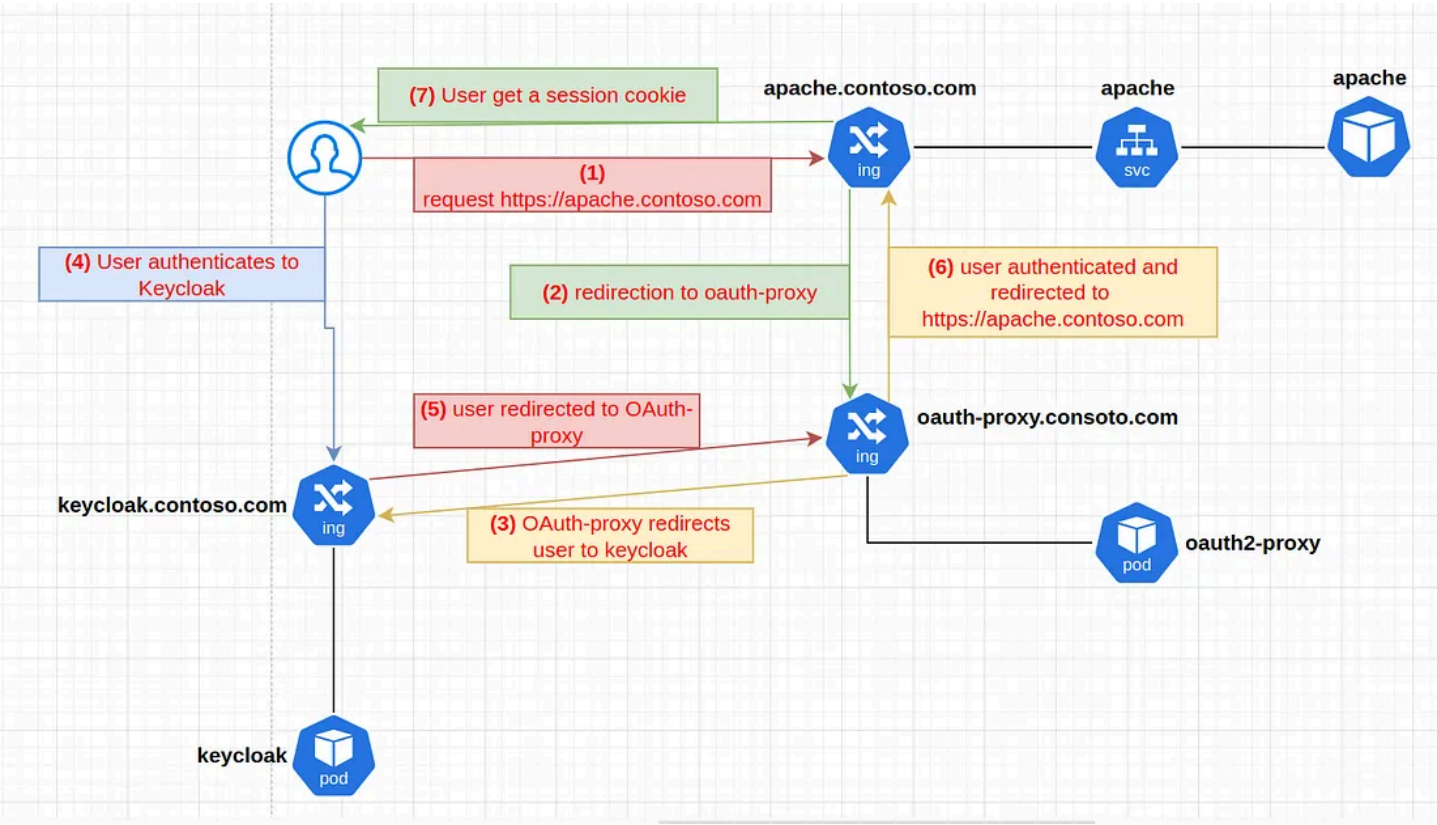
This approach allows to easily integrate authentication with few changes to the existing deployment.

I) Architecture

To realize this PoC, we'll try to add authentication to access a "It works" Apache page.

The Ingress, in front of the Apache Pod/Service, will redirect unauthenticated user to oauth2-proxy which will redirect him to the Keycloak authentication page.

Once authenticated, user will be able to access the "It works" Apache page.



(1) The user try to access to https://apache.contoso.com
(2) The nginx ingress check if the user already authenticated to the OAuth2 Proxy.
(3) The user is not yet authenticated, so OAuth2 Proxy redirect the user to Keycloak
(4) User authenticates to Keycloak
(5) Keycloak redirects user to OAuth2 Proxy
(6) Oauth2 Proxy redirects user to https://apache.contoso.com
(7) The user receive a session cookie which will enable him to authenticate for future requests

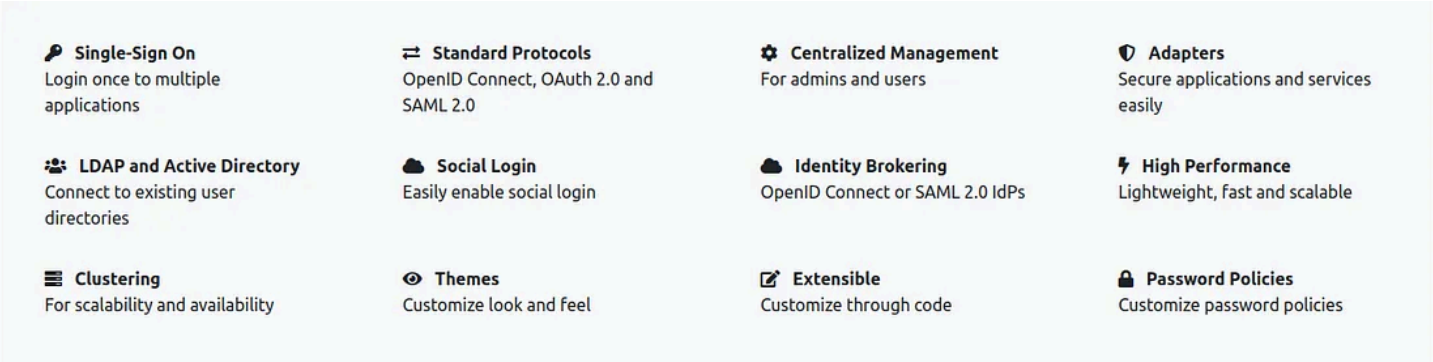
II) Few words about OAuth2 Proxy and Keycloak

OAuth2 Proxy

OAuth2 proxy is a reverse proxy that handles authentication and authorization for web applications using Providers (Google,Keycloak, GitHub,…).

Keycloak

Keycloak is an open-source identity and access management solution, providing authentication, authorization, and user management services for applications and APIs.



https://www.keycloak.org/

III) Deployment

A) Keycloak deployment

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: keycloak-install
  namespace: argocd
spec:
  project: default
  source:
    repoURL: 'registry-1.docker.io/bitnamicharts'
    targetRevision: 21.0.0
    chart: keycloak
    helm:
      values: |
        proxy: edge
        postgresql:
          auth:
            adminUser: **********
            adminPassword: "*****************"
            postgresPassword: "*****************"
            username: bn_keycloak
            password: "**********************"
            database: bitnami_keycloak
          primary:
            persistence:
              storageClass: "longhorn-retain"
              size: 8Gi
          readReplicas:
            persistence:
              storageClass: "longhorn-retain"
              size: 8Gi
  destination:
    server: 'https://kubernetes.default.svc'
    namespace: keycloak
  syncPolicy:
    automated: {}
    syncOptions:
      - CreateNamespace=true
  ignoreDifferences:
    - group: '*'
      kind: '*'
```

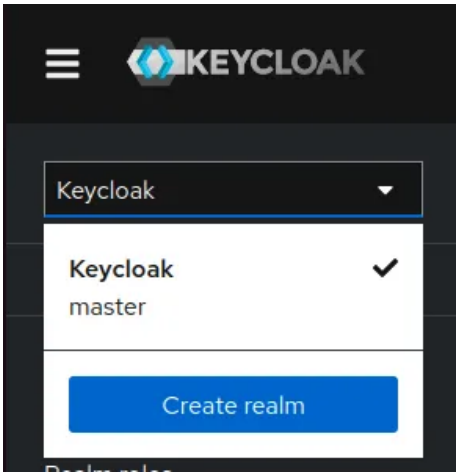
```
        jsonPointers:
          - /metadata/labels/argocd.argoproj.io~1instance
```

- As the password is randomly generated and stored in a secret or PVC, when the chart is upgraded or some changes happens (example: changing *proxy* value) a new random one is generated.
  To avoid this **always set *postgresql.auth.postgresPassword* and *postgresql.auth.password*** (to always have same password used).
- For the PoC purpose, ***proxy*** value has been set to ***edge*** but it is not recommended to use this value in production.

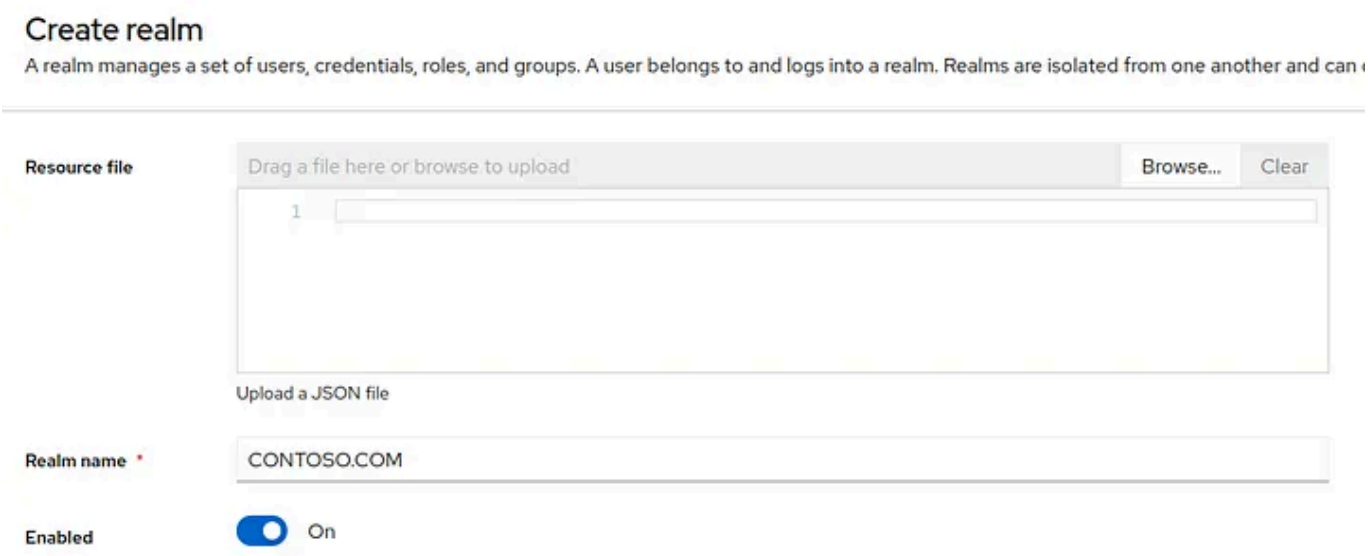
B) Keycloak configuration

1. **Create a realm**

On the top left, click on the list (where "*Keycloak*" is displayed) and select "***Create realm***"

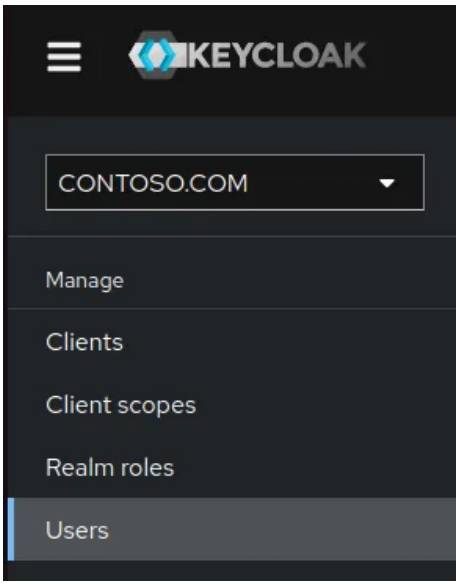


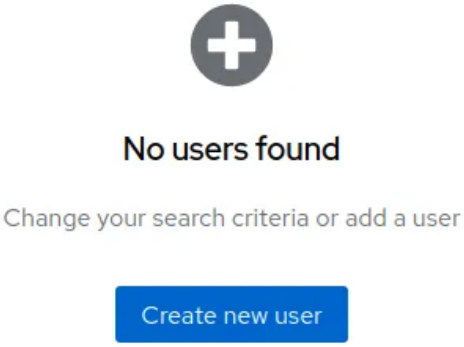
Fill the ***Realm name*** field.



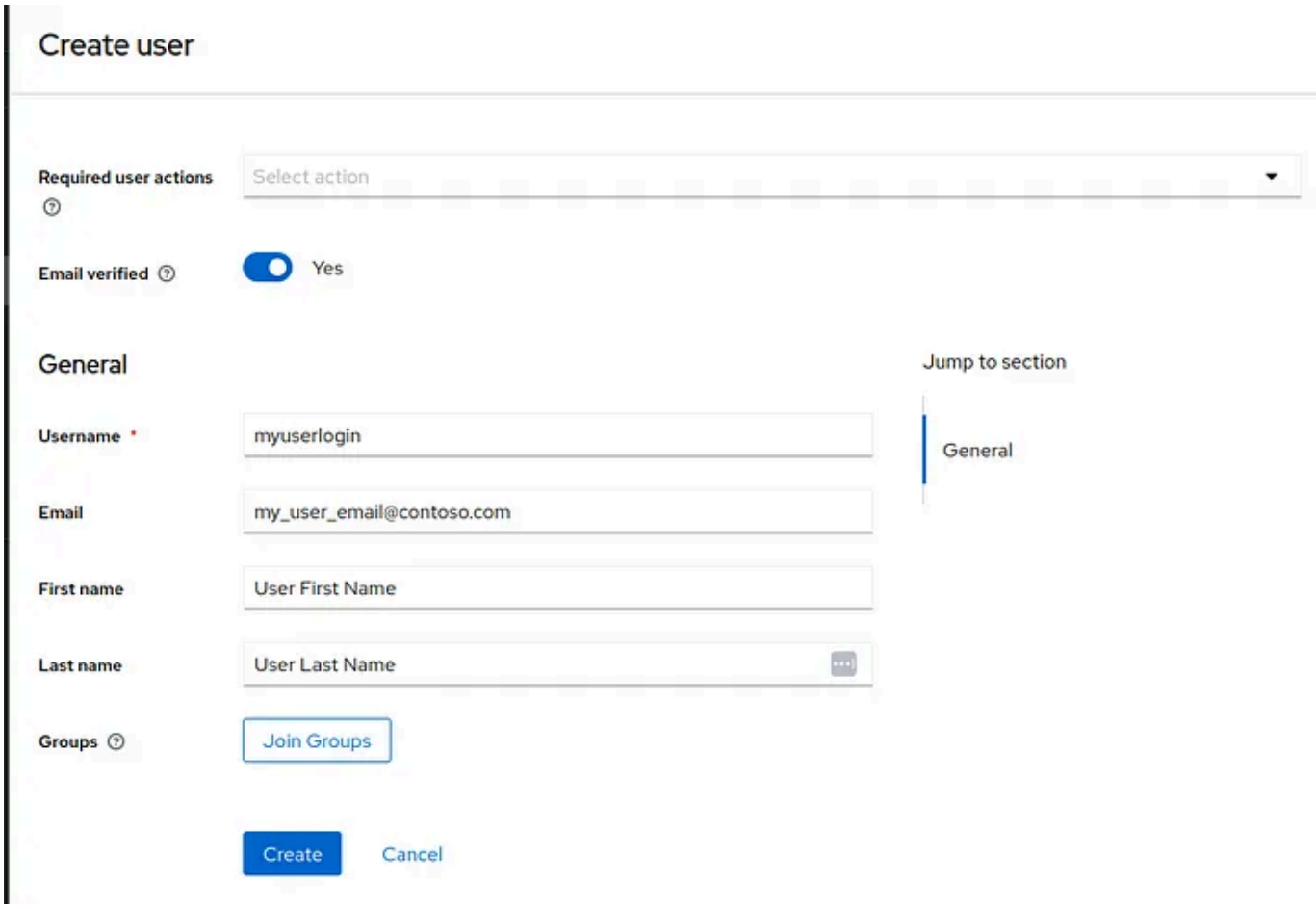
2) Create a new user

In the top left list, select the previously created Realm and go to the ***Users*** section.

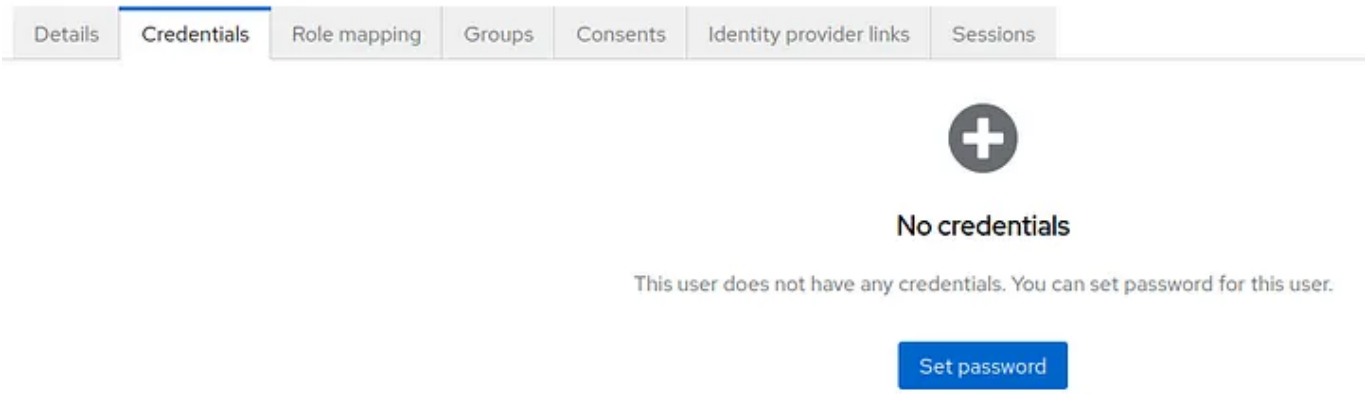


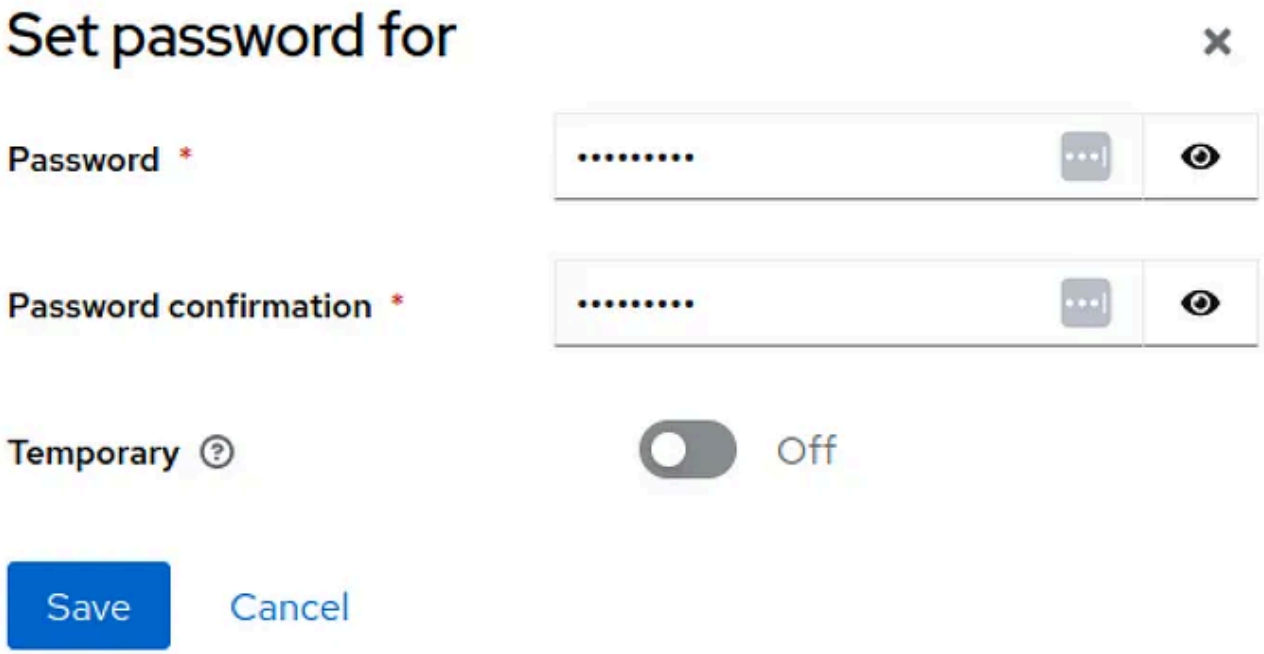


Switch ***Email verified*** to ***Yes*** and fill user information.



Once created, go the user ***Credentials*** tab and set a password.

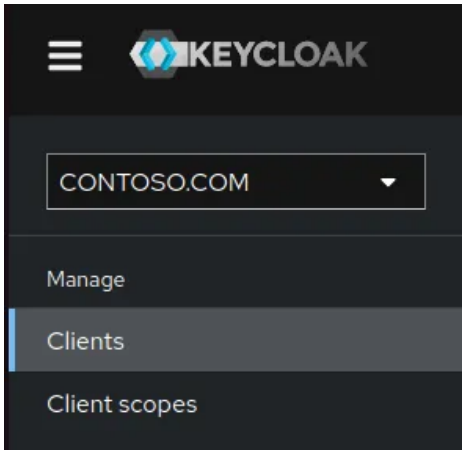




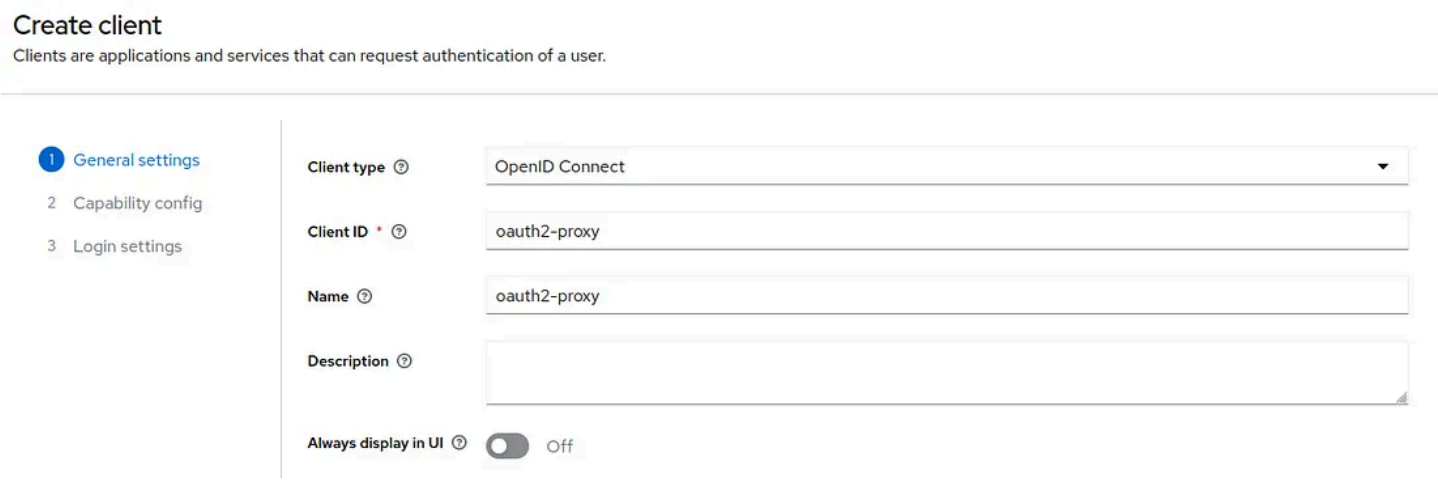
3) Create a new client

We need to create a new OpenID Connect client dedicated to OAuth2 Proxy.

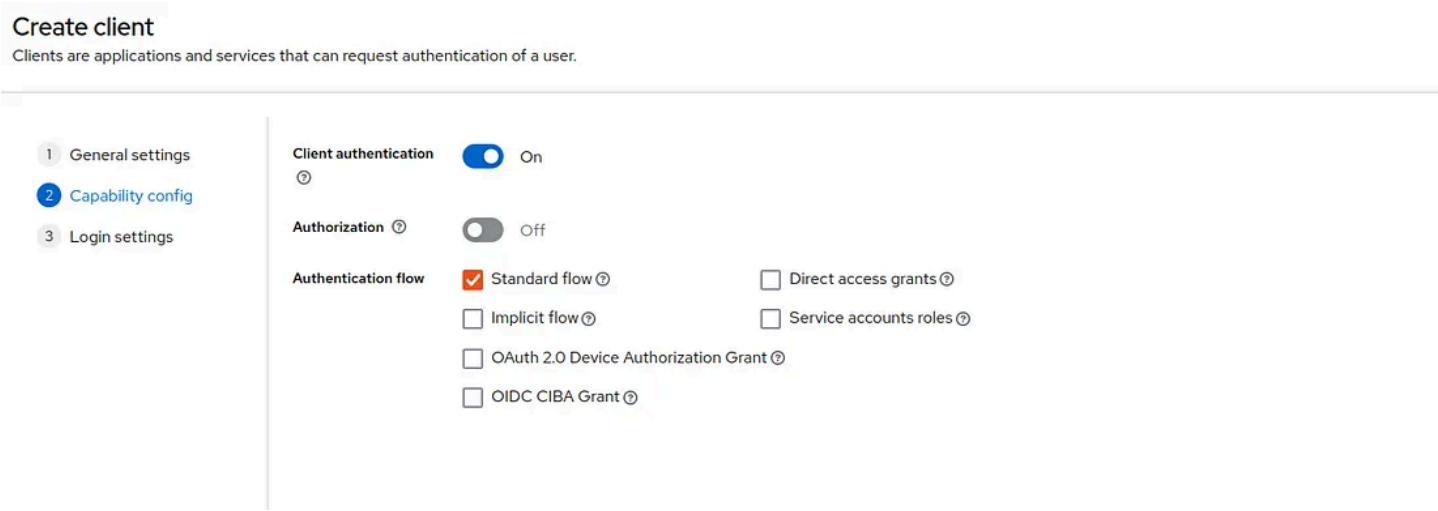
In the top left list, select the previously created Realm and go to the ***Client***s section.



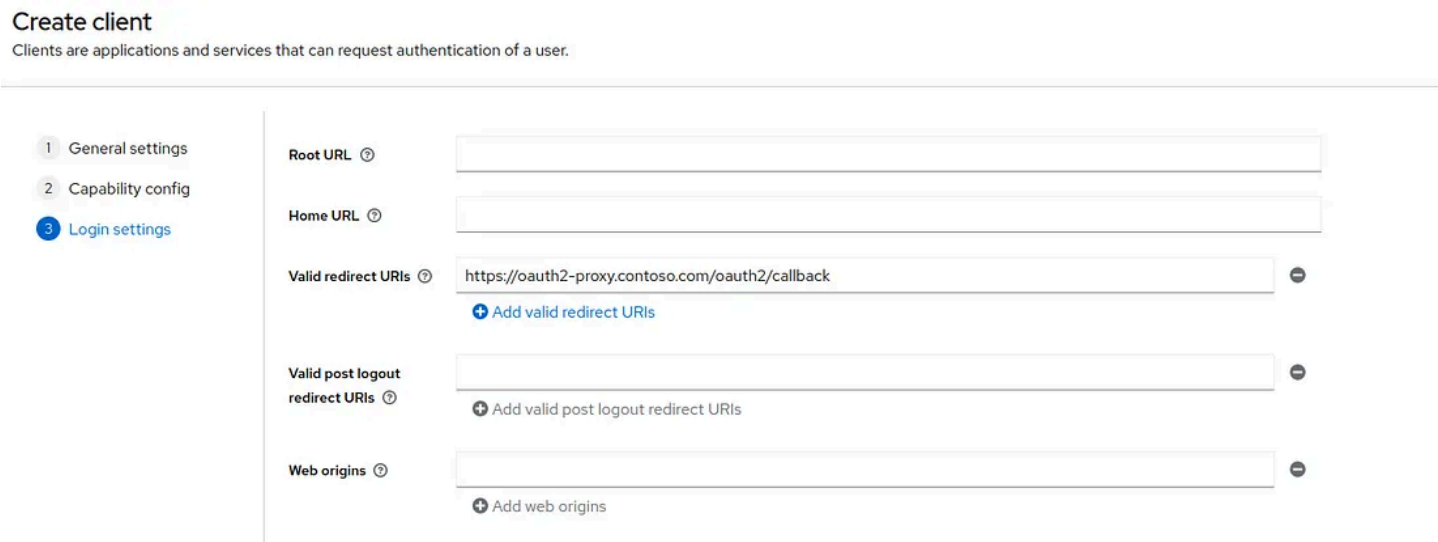
Select ***OpenID Connect*** for ***Client type***.



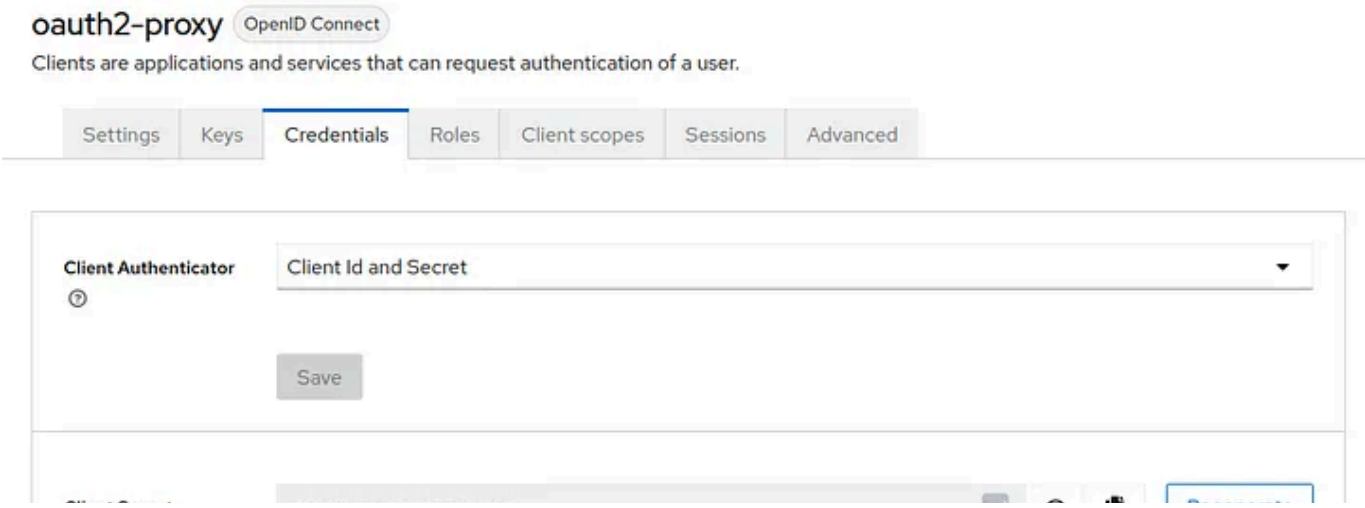
Select "*Standard flow*" and deselect "*Direct access grants*"



Fill "*Valid redirect URIs*" with oauth2-proxy callback URL (in my example: https://oauth2-proxy.contoso.com/oauth2/callback)

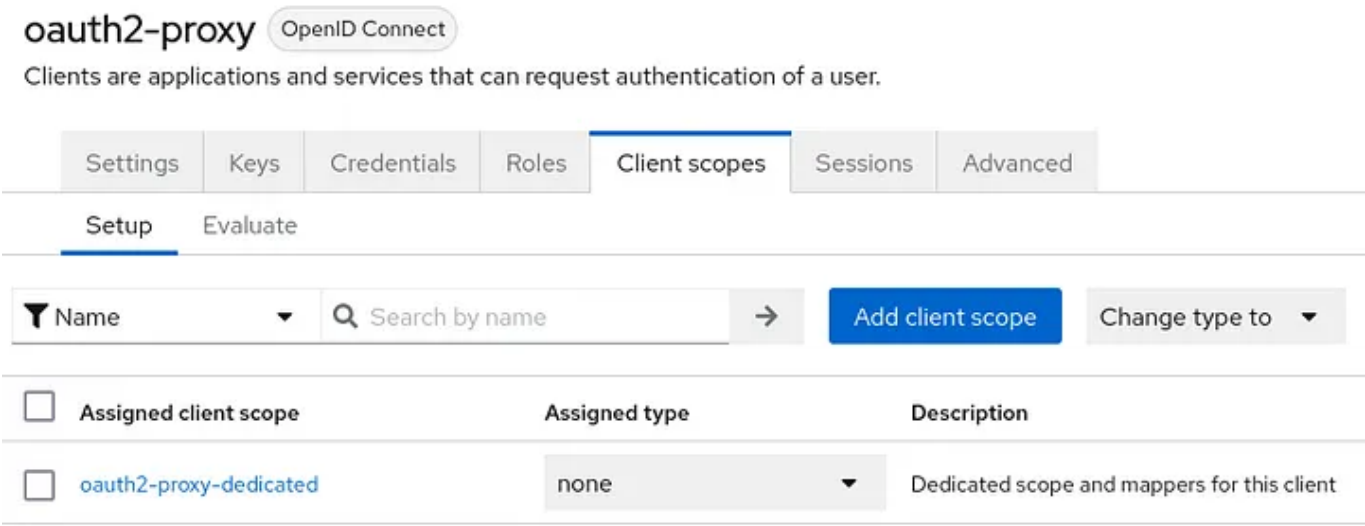


Get client secret from **Credentials** tab

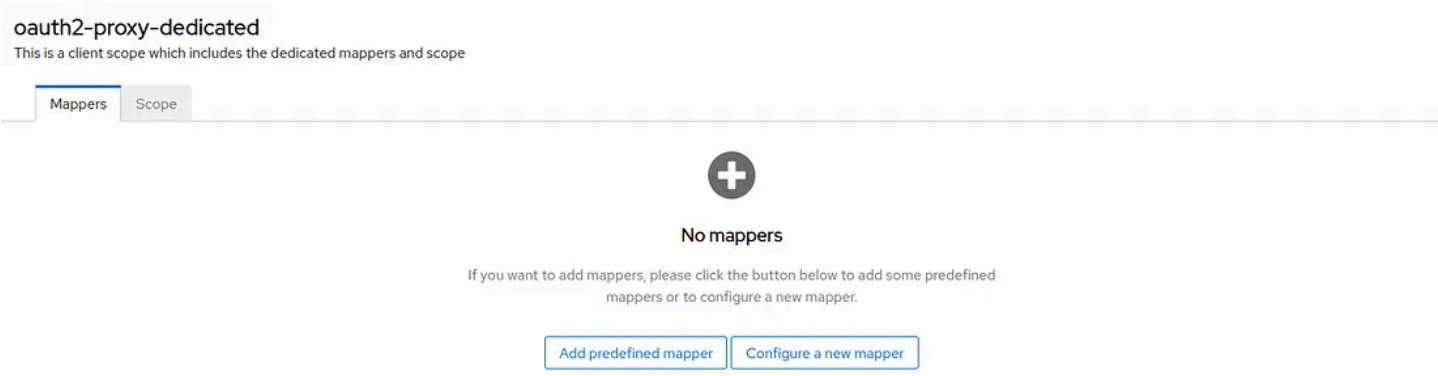


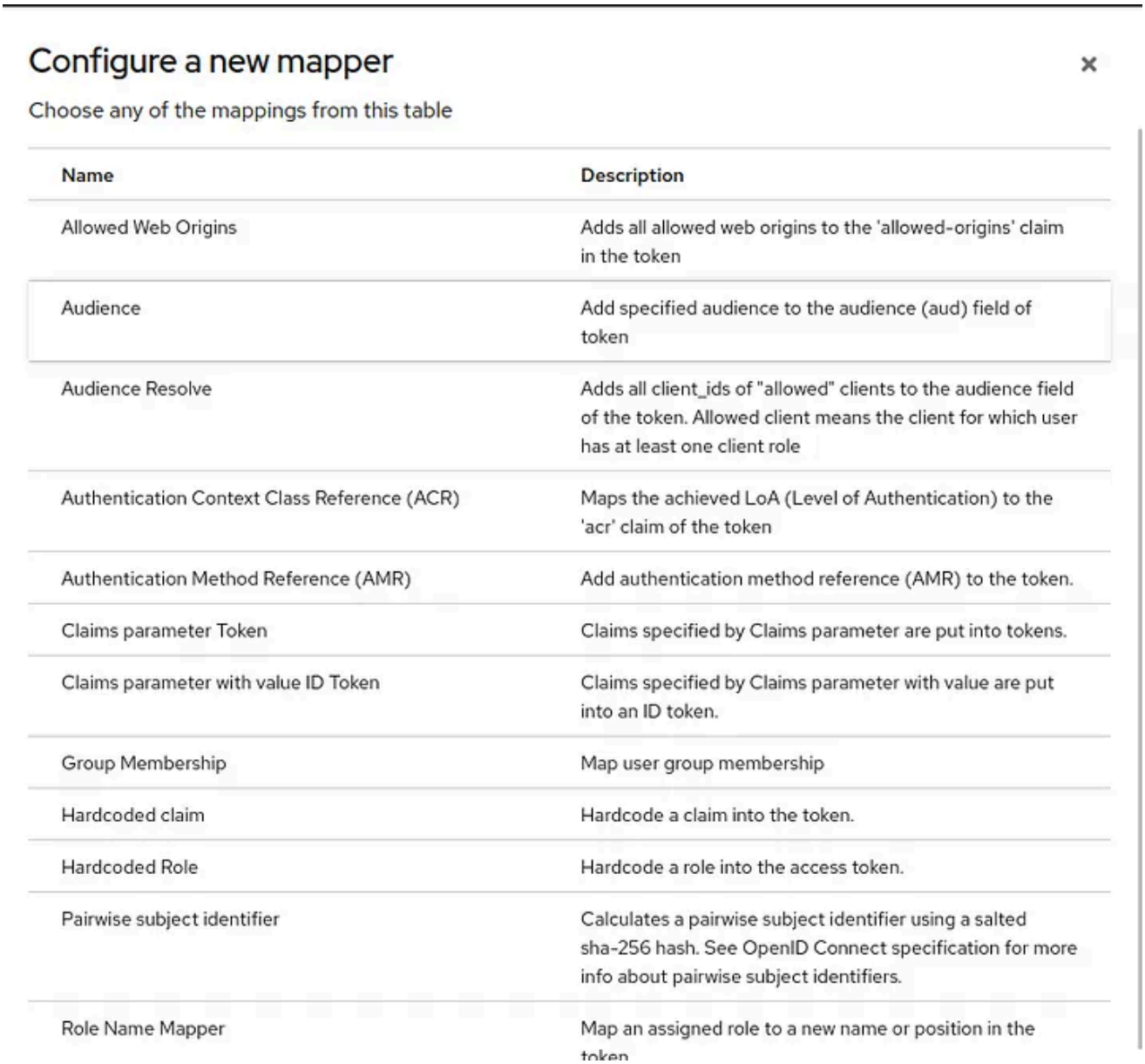
4) Configure a dedicated audience mappers

In ***Clients***, select previously created client then ***Client scopes*** tab



Access the oauth2-proxy dedicated client scope (named *oauth2-proxy-dedicated*) then select "***Configure a new mapper***" and select "***Audience***"



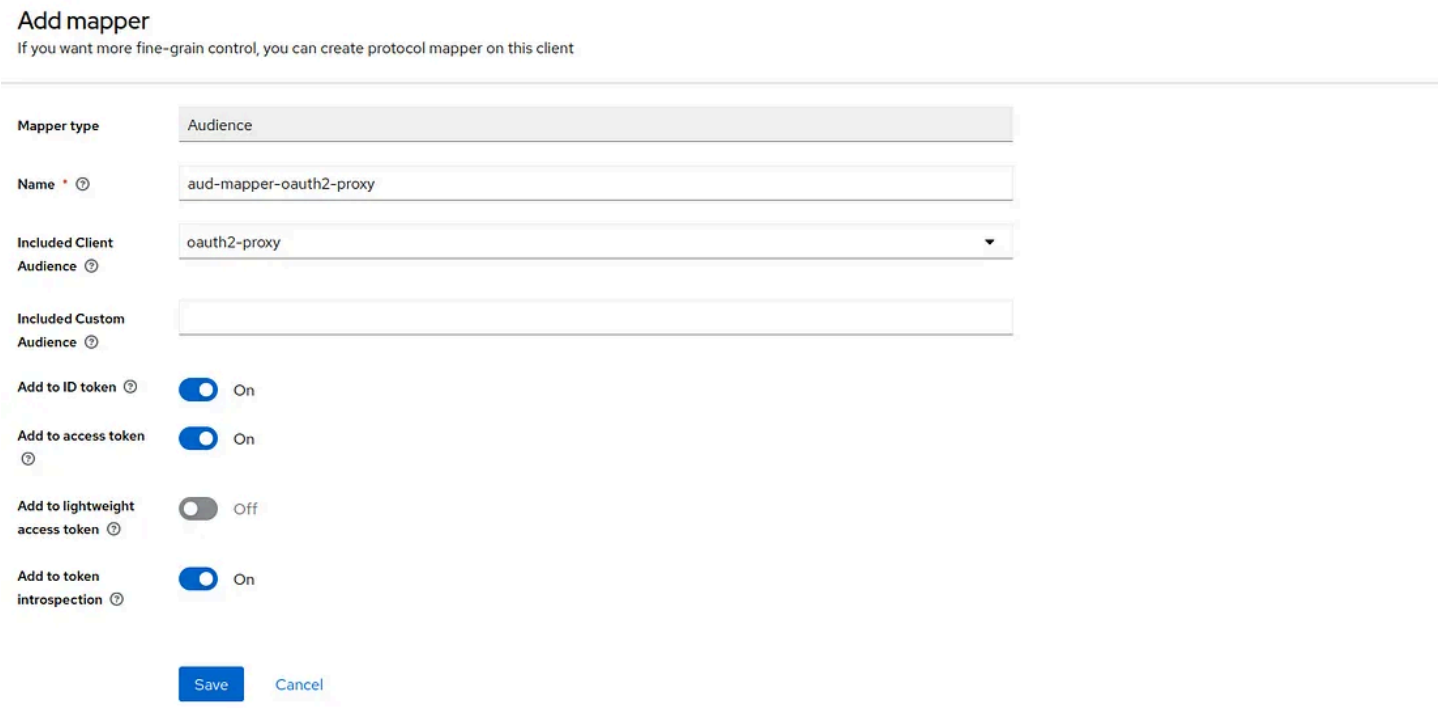


Configure a new mapper

Choose any of the mappings from this table

Name	Description
Allowed Web Origins	Adds all allowed web origins to the 'allowed-origins' claim in the token
Audience	Add specified audience to the audience (aud) field of token
Audience Resolve	Adds all client_ids of "allowed" clients to the audience field of the token. Allowed client means the client for which user has at least one client role
Authentication Context Class Reference (ACR)	Maps the achieved LoA (Level of Authentication) to the 'acr' claim of the token
Authentication Method Reference (AMR)	Add authentication method reference (AMR) to the token.
Claims parameter Token	Claims specified by Claims parameter are put into tokens.
Claims parameter with value ID Token	Claims specified by Claims parameter with value are put into an ID token.
Group Membership	Map user group membership
Hardcoded claim	Hardcode a claim into the token.
Hardcoded Role	Hardcode a role into the access token.
Pairwise subject identifier	Calculates a pairwise subject identifier using a salted sha-256 hash. See OpenID Connect specification for more info about pairwise subject identifiers.
Role Name Mapper	Map an assigned role to a new name or position in the token

Fill name with "***aud-mapper-<client ID>***" (in my example *aud-mapper-oauth2-proxy*), select the client ID in "***Included Client Audience***" list (in my example *oauth2-proxy*) and set "***Add to ID token***" and "***Add to access token***" to "***On***".



C) OAuth2 Proxy deployment

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: oauth2-proxy-install
  namespace: argocd
spec:
  source:
    chart: oauth2-proxy
    repoURL: https://oauth2-proxy.github.io/manifests
    targetRevision: 7.1.0
    helm:
      values: |
        config:
          clientID: oauth2-proxy
          clientSecret: ********************************
          cookieSecret: ********************************
          configFile: |
            provider = "keycloak-oidc"
            provider_display_name = "Keycloak"
            redirect_url = "https://oauth2-proxy.contoso.com/oauth2/callback"
            oidc_issuer_url = "https://keycloak.contoso.com/realms/CONTOSO.COM"
            code_challenge_method = "S256"
            whitelist_domains = ["*.contoso.com"]
            email_domains = ["*"]
            cookie_domains = [".contoso.com"]
            cookie_expire = "1h"
            session_cookie_minimal = "true"
  destination:
    namespace: oauth2-proxy
    server: https://kubernetes.default.svc
  project: default
  syncPolicy:
    automated: {}
    syncOptions:
      - CreateNamespace=true
```

- *clientID* is the client ID value set in Keycloak
- *clientSecret* is the client secret previously get in Keycloak

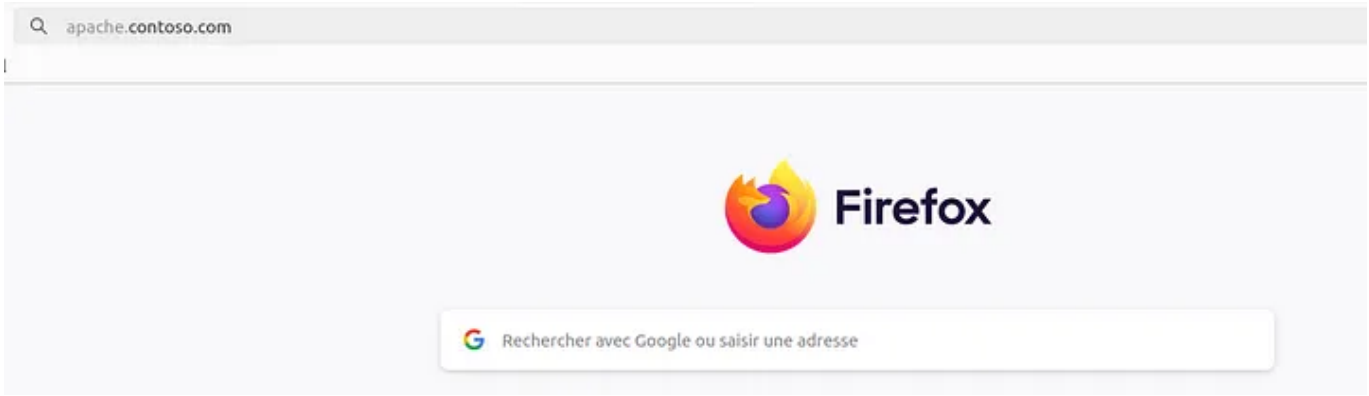
D) Nginx ingress configuration

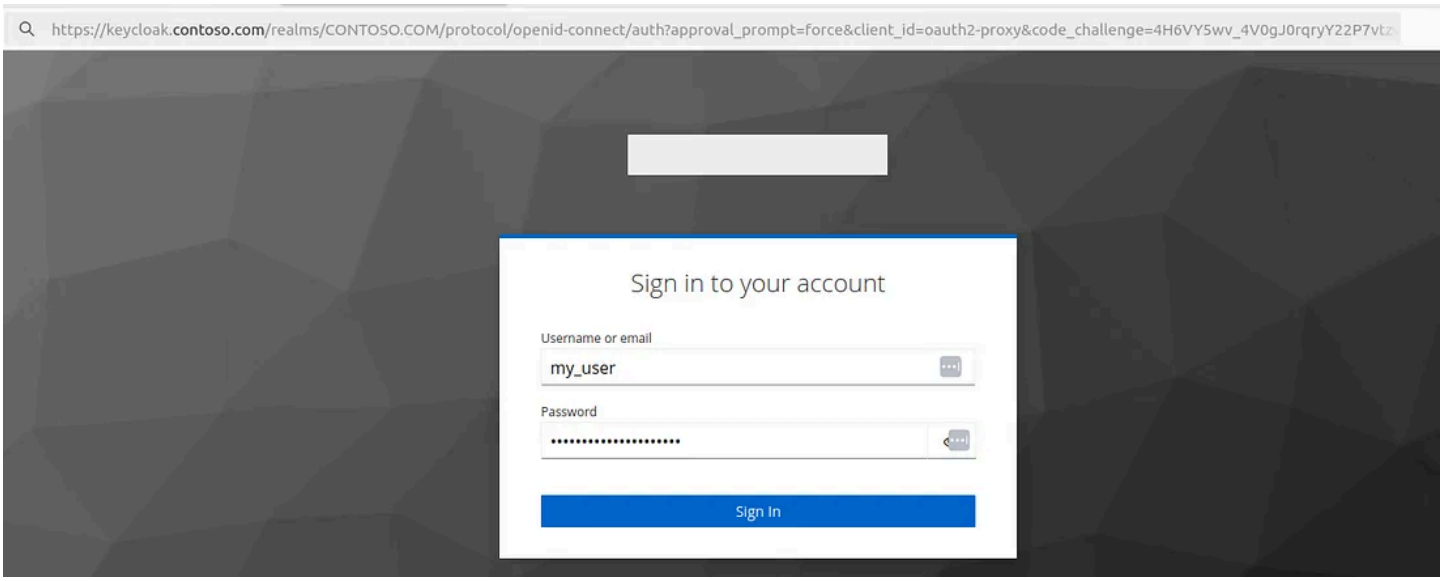
We need to add 2 annotations to the Ingress Apache frontend to set authentication requirement.

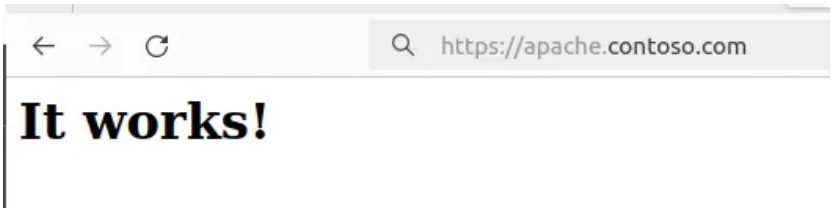
```
annotations:
  nginx.ingress.kubernetes.io/auth-url: "https://oauth2-proxy.contoso.com/oauth2/auth"
  nginx.ingress.kubernetes.io/auth-signin: "https://oauth2-proxy.contoso.com/oauth2/start"
```

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: apache-ingress
  namespace: apache
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
    nginx.ingress.kubernetes.io/ssl-passthrough: "false"
    nginx.ingress.kubernetes.io/auth-url: "https://oauth2-proxy.contoso.com/oauth2/auth"
    nginx.ingress.kubernetes.io/auth-signin: "https://oauth2-proxy.contoso.com/oauth2/start"
spec:
  ingressClassName: nginx
  rules:
  - host: apache.contoso.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: apache
            port:
              number: 80
  tls:
  - hosts:
    - apache.contoso.com
    secretName: apache-certificate
```

E) Let's try it!







Hope you enjoyed!

Oauth2 Proxy | Keycloak | Sso | Oauth2 | Oauth2 Provider

Follow

Written by Genesta Sebastien

83 Followers

More from Genesta Sebastien