Projet : Naldeo Industrial Metrics

Objet : Problème d'authentification automatique à Grafana

Solution : Utilisation système d'authentification externe, tel que OpenID Connect ou OAuth, pour gérer l'authentification et l'autorisation.

Table des matières

1. Configurer le fournisseur OpenID Connect ou OAuth pour gérer l'authentification.
2. Dans l'application Angular, créer un formulaire d'identification qui envoie une demande d'authentification au fournisseur OpenID Connect ou OAuth.
3. Après avoir reçu une réponse réussie du fournisseur OpenID Connect ou OAuth, stocker le jeton d'accès dans l'application Angular.
4. Dans l'application Angular, effectuer des appels API à Grafana, en passant le jeton d'accès dans l'en-tête d'autorisation de chaque requête sous forme de jeton porteur.
5. Sur le serveur Grafana, configurer une URL de point d'API pour valider le jeton d'accès et accorder ou refuser l'accès à l'API.

1- **Configurer Grafana Docker pour utiliser un système d'authentification externe, tel que OpenID Connect ou OAuth**

Pour configurer Grafana Docker pour utiliser un système d'authentification externe tel qu'OpenID Connect ou OAuth, vous pouvez utiliser les étapes suivantes comme guide:

a- Ajouter les variables d'environnements dans le docker-compose:

```
GF_AUTH_OPENID_ENABLED=true
GF_AUTH_OPENID_ISSuer=https://your-openid-provider.com
GF_AUTH_OPENID_CLIENT_ID=your-client-id
GF_AUTH_OPENID_CLIENT_SECRET=your-client-secret
```

b- Démarrez le conteneur Docker Grafana.
c- Installez le plugin Grafana OAuth ou OpenID Connect, en fonction du système d'authentification que vous utilisez, à l'intérieur du conteneur Docker. Vous pouvez le faire en utilisant la CLI Grafana ou l'interface utilisateur Grafana.
d- Dans l'interface utilisateur Grafana, accédez à la page Configuration > Authentication.
e- Dans la section fournisseurs d'authentification, sélectionnez le plugin OAuth ou OpenID Connect que vous avez installé.
f- Configurez le fournisseur d'authentification en fournissant les informations nécessaires, telles que l'ID client, le secret client, le point de terminaison d'autorisation, le point de terminaison de jeton et le point de terminaison de profil d'utilisateur. Les informations spécifiques requises dépendent du système d'authentification que vous utilisez et du fournisseur OAuth ou OpenID Connect que vous utilisez.
g- Enregistrez les modifications et redémarrez le conteneur Docker Grafana.
h- Connectez-vous à Grafana avec vos identifiants OpenID Connect ou OAuth pour confirmer que le système d'authentification externe fonctionne correctement.

2- **Dans l'application Angular, créer un formulaire d'identification qui envoie une demande d'authentification au fournisseur OpenID Connect ou OAuth.**

a- Installez la bibliothèque "oidc-client" qui est une bibliothèque JavaScript qui vous permet d'intégrer facilement OpenID Connect à votre application Angular.

b- Définissez les détails de votre fournisseur d'identité dans le fichier de configuration, tels que le client_id, l'URL d'autorisation, l'URL de token, etc.
c- Créez un composant Angular pour gérer la soumission du formulaire d'identification. Ce composant utilisera la bibliothèque "oidc-client" pour envoyer une demande d'authentification au fournisseur.
d- Créez un formulaire HTML dans ce composant avec les champs nécessaires pour entrer les informations d'identification de l'utilisateur.
e- Lorsque le formulaire est soumis, appelez la méthode "signIn" de la bibliothèque "oidc-client" pour envoyer une demande d'authentification au fournisseur.
f- En cas de réussite de l'authentification, redirigez l'utilisateur vers la page souhaitée ou affichez les informations d'identification dans le composant

Voici un exemple de code:

```
import { Component } from '@angular/core';
import { UserManager, User } from 'oidc-client';

@Component({
  selector: 'app-login',
  template: `
    <form (ngSubmit)="onSubmit()">
      <div>
        <label for="username">Username:</label>
        <input type="text" id="username" [(ngModel)]="username">
      </div>
      <div>
        <label for="password">Password:</label>
        <input type="password" id="password" [(ngModel)]="password">
      </div>
      <button type="submit">Login</button>
    </form>
  `
})
```

```
export class LoginComponent {
  private manager = new UserManager({
    authority: 'https://your-openid-provider.com',
    client_id: 'your-client-id',
    redirect_uri: 'http://localhost:4200/callback',
    response_type: 'code',
    scope: 'openid profile email',
    post_logout_redirect_uri: 'http://localhost:4200/',
  });

  username: string;
  password: string;

  onSubmit() {
    this.manager.signinRedirect({
      username: this.username,
      password: this.password
    }).then(function () {
      console.log('signinRedirect done');
    }).catch(function (err) {
      console.log(err);
    });
  }
}
```

3- **Après avoir reçu une réponse réussie du fournisseur OpenID Connect ou OAuth, vous pouvez stocker le jeton d'accès dans l'application Angular de la façon suivante :**

a- Définissez un composant qui sera utilisé pour gérer la réception de la réponse d'authentification. Ce composant peut être appelé "CallbackComponent".
b- Configurez votre application Angular pour rediriger vers ce composant lorsqu'une réponse d'authentification est reçue. Cela peut être fait en utilisant une route correspondante dans le fichier de configuration de votre application.
c- Dans le composant "CallbackComponent", utilisez la méthode "signinCallback" de la bibliothèque "oidc-client" pour traiter la réponse d'authentification. Cette méthode vous renverra un objet "User" qui contiendra le jeton d'accès.
d- Stocker le jeton d'accès dans le stockage local du navigateur pour pouvoir y accéder plus tard. Vous pouvez utiliser le stockage local HTML5 pour cela.

Voici un exemple de code :

```
export class CallbackComponent {
  private manager = new UserManager({
    authority: 'https://your-openid-provider.com',
    client_id: 'your-client-id',
    redirect_uri: 'http://localhost:4200/callback',
    response_type: 'code',
    scope: 'openid profile email',
    post_logout_redirect_uri: 'http://localhost:4200/',
  });

  ngOnInit() {
    this.manager.signinRedirectCallback().then((user: User) => {
      localStorage.setItem('access_token', user.access_token);
      console.log('signinRedirectCallback successful, access token stored');
    }).catch((err) => {
      console.log(err);
    });
  }
}
```

4- **Dans l'application Angular, effectuer des appels API à Grafana, en passant le jeton d'accès dans l'en-tête d'autorisation de chaque requête sous forme de jeton porteur**.

```
import { HttpClient, HttpHeaders } from '@angular/common/http';

...

constructor(private http: HttpClient) {}

...

private accessToken = 'your_access_token';

...

makeApiCallToGrafana() {
  const headers = new HttpHeaders({
    'Authorization': 'Bearer ' + this.accessToken
  });
  this.http.get('https://grafana.com/api/', { headers }).subscribe(
    data => {
      console.log(data);
    },
    error => {
      console.error(error);
    }
  );
}
```

5- **Sur le serveur Grafana, configurer une URL de point d'API pour valider le jeton d'accès et accorder ou refuser l'accès à l'API.**

La configuration d'une URL de point d'API pour valider le jeton d'accès et accorder ou refuser l'accès à l'API dépend de la méthode d'authentification que vous utilisez pour votre serveur Grafana.

Si vous utilisez l'authentification par jeton, vous pouvez configurer une URL de point d'API en utilisant un middleware pour valider le jeton d'accès dans l'en-tête d'autorisation de la requête. Par exemple, si vous utilisez Express.js comme framework de serveur, vous pouvez ajouter le code suivant pour valider le jeton d'accès :

```
const jwt = require('jsonwebtoken');

...

const secretKey = 'your_secret_key';

...

app.use((req, res, next) => {
  const authorizationHeader = req.headers.authorization;
  if (!authorizationHeader) {
    return res.status(401).send({ error: 'Unauthorized' });
  }
  const token = authorizationHeader.split(' ')[1];
  try {
    const decoded = jwt.verify(token, secretKey);
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(401).send({ error: 'Unauthorized' });
  }
});

...

app.get('/api', (req, res) => {
  // Your API logic here
});
```