Authentication

Keycloak

Keycloak is an open-source Identity and Access Management (IAM) tool by Red Hat that enables secure authentication and authorization. It simplifies access with Single Sign-On (SSO), letting users log in once to access multiple applications. Supporting protocols like OpenID Connect and SAML, it integrates easily with various systems. We use Keycloak because it provides us with:

- Single Sign-On (SSO): One login for multiple apps, improving user experience.
- Flexible Integration: Supports various protocols and existing identity providers.
- Centralized Management: Unified platform for managing users, permissions, and sessions.
- Open Source: Customizable and extensible, with community-driven support.

Keycloak with Jupyterhub

Integrating Keycloak with JupyterHub allows for robust authentication using Keycloak's identity management features. This integration can be achieved using the GenericOAuthenticator class, which facilitates the OpenID Connect (OIDC) protocol. By configuring JupyterHub to authenticate users through Keycloak, you can centralize user management and leverage Keycloak's capabilities such as group and role assignments.

Here’s an example entry for a config.yaml file to set up Keycloak authentication with JupyterHub:

```
 hub:
   config:
     JupyterHub:
        authenticator_class: generic-oauth
     GenericOAuthenticator:
        client_id: "your-client-id"
        client_secret: "your-client-secret"
        oauth_callback_url:
"https://your-jupyterhub-domain/hub/oauth_callback"
        authorize_url:
"https://keycloak-host/auth/realms/your-realm/protocol/openid-connect/auth"
        token_url:
"https://keycloak-host/auth/realms/your-realm/protocol/openid-connect/token"
        userdata_url:
"https://keycloak-host/auth/realms/your-realm/protocol/openid-connect/userin
fo"
        username_key: "preferred_username"  # or another key based on your
Keycloak setup
        login_service: "Keycloak"
        scope:
           - "openid"
           - "profile"
           - "groups"
```

where:

- **client_id** and **client_secret**: These are obtained from your Keycloak client settings.
- **oauth_callback_url**: This URL should match the redirect URI configured in Keycloak for your application.
- **authorize_url**, **token_url**, and **userdata_url**: These endpoints are part of the OIDC specification and are used for user authentication and information retrieval.
- **username_key**: This specifies which claim from the Keycloak token should be used as the username in JupyterHub.
- **scope**: This defines the permissions being requested, including access to user profile information and group memberships.

This configuration enables JupyterHub to authenticate users against Keycloak, providing a seamless login experience while managing user roles and permissions effectively.

Config.yaml for JupyterHub Helm Chart

The `config.yaml` file is essential for deploying JupyterHub via Helm charts on Kubernetes. It allows customization of the deployment to suit specific requirements by overriding the default values provided by the chart.

Key Sections of config.yaml

Hub Configuration

Configures the core JupyterHub settings, such as the hub's base URL and internal networking options:

```
hub:
  baseUrl: /hub/
```

Authenticator Settings

Defines the authentication mechanism (e.g., OAuth, LDAP, or GitHub):

```
JupyterHub:
      authenticator_class: generic-oauth
   GenericOAuthenticator:
      client_id: "your-client-id"
      client_secret: "your-client-secret"
      oauth_callback_url:
"https://your-jupyterhub-domain/hub/oauth_callback"
      authorize_url:
"https://keycloak-host/auth/realms/your-realm/protocol/openid-connect/auth"
      token_url:
"https://keycloak-host/auth/realms/your-realm/protocol/openid-connect/token"
```

```
      userdata_url:
"https://keycloak-host/auth/realms/your-realm/protocol/openid-connect/userin
fo"
      username_key: "preferred_username"  # or another key based on your
Keycloak setup
      login_service: "Keycloak"
      scope:
         - "openid"
         - "profile"
         - "groups"
```

User Environment

Specifies the user image, resource limits, and environment variables for the notebook server:

```
singleuser:
  image:
    name: jupyter/base-notebook
    tag: latest
  cpu:
    limit: 1
  memory:
    limit: 1Gi
```

Storage Options

Configures persistent storage for user data:

```
storage:
  type: pvc
  capacity: 10Gi
  storageClass: standard
```

Ingress and Networking

Defines secure communications and networking setups:

```
ingress:
 enabled: true
 annotations:
   kubernetes.io/ingress.class: nginx
   nginx.ingress.kubernetes.io/proxy-body-size: 500m
 ingressClassName: "public"
 hosts:
   - hub.eolab.de
 pathSuffix:
 pathType: Prefix
```

```
  tls: []
```

User Profiles

Supports multiple profiles with unique configurations for different user needs:

```
singleuser:
  profiles:
    - display_name: "Standard User"
      description: "1 CPU, 1GB RAM"
      kubespawner_override:
        cpu_limit: 1
        mem_limit: 1Gi
    - display_name: "Power User"
      description: "4 CPU, 8GB RAM"
      kubespawner_override:
        cpu_limit: 4
        mem_limit: 8Gi
```

Best Practices

- **Version Control**: Track changes to `config.yaml` using version control.
- **Secret Management**: Use Kubernetes secrets instead of hardcoding sensitive data.
- **Testing Before Deployment**: Validate the configuration locally using `helm template`:

The `config.yaml` file is versatile and enables tailored JupyterHub deployments, making it a cornerstone for scalable, secure, and user-friendly setups on Kubernetes.

Config.yaml

Following is the config.yaml used in the Crunchy Cloud setup:

```
hub:
  revisionHistoryLimit:
  db:
    pvc:
      storageClassName: nfs-csi
  config:
    GenericOAuthenticator:
      client_id: jupyter-hub
      client_secret: <client-secret>
      oauth_callback_url: http://hub.eolab.de/hub/oauth_callback
      authorize_url:
https://auth.eolab.de/auth/realms/lab3/protocol/openid-connect/auth
      token_url:
```

```
https://auth.eolab.de/auth/realms/lab3/protocol/openid-connect/token
      userdata_url:
https://auth.eolab.de/auth/realms/lab3/protocol/openid-connect/userinfo
      login_service: keycloak
      username_key: preferred_username
      userdata_params:
        state: state
      # In order to use keycloak client's roles as authorization layer
      claim_groups_key: roles
      #allowed_groups:
      #  - user
      #admin_groups:
       #   - admin
      allow_all: True
    JupyterHub:
      authenticator_class: generic-oauth
      bind_url: http://127.0.0.1:8000
# singleuser relates to the configuration of KubeSpawner which runs in the
hub
# pod, and its spawning of user pods such as jupyter-myusername.
singleuser:
  storage:
    type: dynamic
    capacity: 5Gi
    homeMountPath: /home/jovyan
    dynamic:
      storageClass: nfs-csi
      pvcNameTemplate: claim-{username}{servername}
      volumeNameTemplate: volume-{username}{servername}
      storageAccessModes: [ReadWriteOnce]
  image:
    name: jupyter/scipy-notebook
    tag: "latest"
  profileList:
    - display_name: "Python Environment"
      description: "Python environment with all packages for Scientific
Programming and Data Analysis"
      default: true
    - display_name: "R Environment"
      description: "R development environment for Statistical and Data
Analysis"
      kubespawner_override:
        image: jupyter/r-notebook:latest
    - display_name: "Datascience Environment"
      description: "Datascience environment with Python, R and Julia"
      kubespawner_override:
        image: jupyter/datascience-notebook:latest
  cpu:
    limit:
    guarantee:
  memory:
```

```
    limit:
    guarantee: 1G
  extraResource:
    limits: {}
    guarantees: {}
  cmd: jupyterhub-singleuser
  defaultUrl: /lab/

# scheduling relates to the user-scheduler pods and user-placeholder pods.
scheduling:
  userScheduler:
    enabled: false

ingress:
  enabled: true
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/proxy-body-size: 500m
  ingressClassName: "public"
  hosts:
    - hub.eolab.de
  pathSuffix:
  pathType: Prefix
  tls: []

# cull relates to the jupyterhub-idle-culler service, responsible for
evicting
# inactive singleuser pods.
#
# The configuration below, except for enabled, corresponds to command-line
flags
# for jupyterhub-idle-culler as documented here:
#
https://github.com/jupyterhub/jupyterhub-idle-culler#as-a-standalone-script
#
cull:
  enabled: true
  users: false # --cull-users
  adminUsers: true # --cull-admin-users
  removeNamedServers: false # --remove-named-servers
  timeout: 3600 # --timeout
  every: 600 # --cull-every
  concurrency: 10 # --concurrency
  maxAge: 0 # --max-age

debug:
  enabled: false

global:
  safeToShowValues: false
```