

Keycloak Deployment

Pre Requisites

- A running Kubernetes Cluster - either EKS or Minikube (**Note:** Minikube may cause some issues with port forwarding)
- Kubectl installed
- Helm installed
- Docker/Docker Desktop for Minikube
- An EBS Storage Class (AWS)

Overview

This configuration will deploy:

- a keycloak statefulset
- a postgresql statefulset and persistent volume

Steps

The following steps should be carried out within your Kubernetes environment, once authenticated to the cluster

1. Add the Bitnami Helm Repo

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

2. Create a new directory and add a values.yaml file

```
mkdir keycloak
cd keycloak
```

3. Customise the values.yaml file for initial key cloak deployment

Note: This will also pull default values from the bitnami/keycloak helm values. Additionally, you can configure more custom users.

```
global:
  storageClass: gp2
tls:
  enabled: true
  autoGenerated: true
auth:
  adminPassword: Test
postgresql:
  auth:
    password: test
    postgresPassword: test
```

4. Before deploying the helm chart, create a namespace in the cluster

```
kubectl create ns keycloak
```

5. Install the keycloak helm chart with custom values file in the new namespace.

Note: Make sure you are in the correct directory with the values.yaml file

```
helm upgrade --install keycloak bitnami/keycloak --values values.yaml -n keycloak
```

Instead of creating the values.yaml file, you could also declare all of these values in the helm install command e.g.

```
helm upgrade --install keycloak bitnami/keycloak --set tls.enabled=true --set tls.autoGenerated=true --set auth.adminPassword=Test --set po
```

6. Confirm that the deployment has been successful and check the logs. You should see a keycloak pod and a keycloak-postgresql pod running.

```
kubectl get pods -n keycloak
kubectl logs <name-of-pod> -n keycloak -f
```

Example of successful logs:

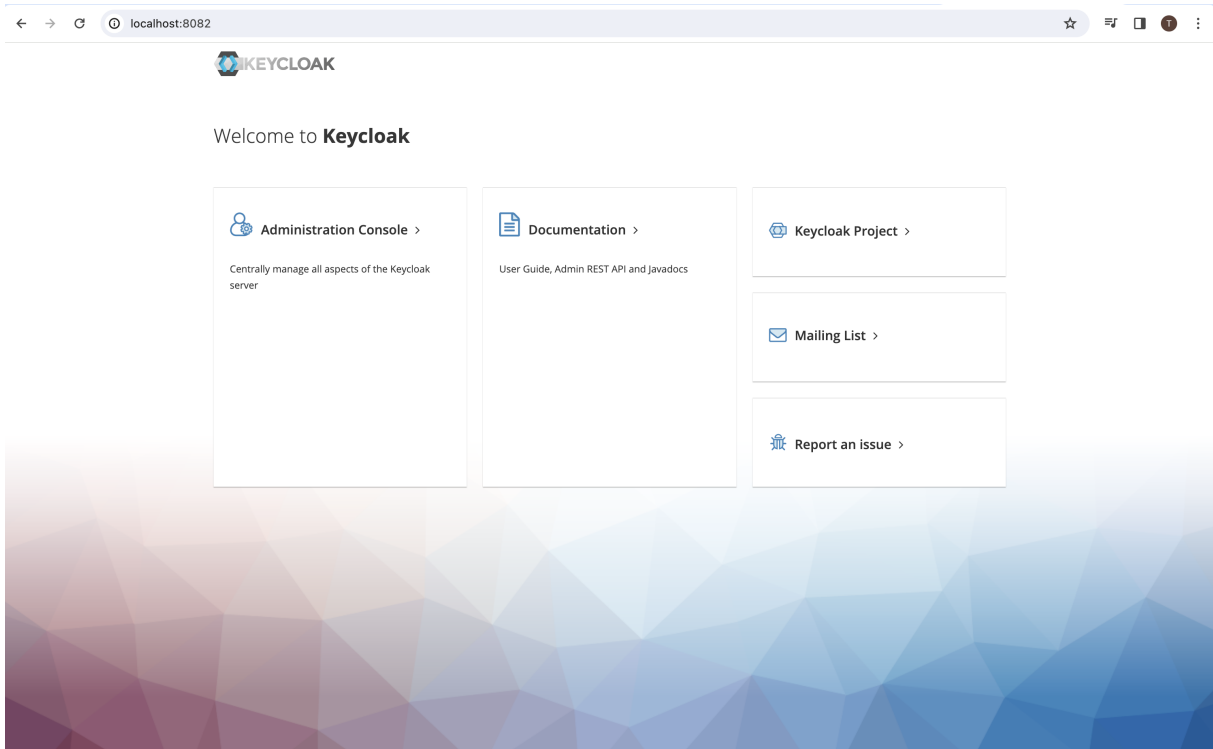
```
postgresql 10:00:21.38
postgresql 10:00:21.38 Welcome to the Bitnami postgresql container
postgresql 10:00:21.38 Subscribe to project updates by watching https://github.com/bitnami/bitnami-docker-postgresql
postgresql 10:00:21.39 Submit issues and feature requests at https://github.com/bitnami/bitnami-docker-postgresql/issues
postgresql 10:00:21.39
postgresql 10:00:21.39 DEBUG ==> Configuring libnss_wrapper...
postgresql 10:00:21.40 INFO ==> ** Starting PostgreSQL setup **
postgresql 10:00:21.45 INFO ==> Validating settings in POSTGRESQL_* env vars..
postgresql 10:00:21.46 INFO ==> Loading custom pre-init scripts...
postgresql 10:00:21.46 INFO ==> Initializing PostgreSQL database...
postgresql 10:00:21.47 DEBUG ==> Ensuring expected directories/files exist...
postgresql 10:00:21.55 INFO ==> pg_hba.conf file not detected. Generating it...
postgresql 10:00:21.55 INFO ==> Generating local authentication configuration
postgresql 10:00:21.56 INFO ==> Deploying PostgreSQL with persisted data...
postgresql 10:00:21.57 INFO ==> Configuring replication parameters
postgresql 10:00:21.65 INFO ==> Configuring fsync
postgresql 10:00:21.74 INFO ==> Loading custom scripts...
postgresql 10:00:21.75 INFO ==> Enabling remote connections
postgresql 10:00:21.77 INFO ==> ** PostgreSQL setup finished! **

postgresql 10:00:21.78 INFO ==> ** Starting PostgreSQL **
2023-09-14 10:00:21.873 GMT [1] LOG:  pgaudit extension initialized
2023-09-14 10:00:21.873 GMT [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2023-09-14 10:00:21.873 GMT [1] LOG:  listening on IPv6 address ":::", port 5432
2023-09-14 10:00:21.880 GMT [1] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
2023-09-14 10:00:22.096 GMT [92] LOG:  database system was shut down at 2023-09-13 21:27:37 GMT
2023-09-14 10:00:22.309 GMT [1] LOG:  database system is ready to accept connections
2023-09-14 10:12:44.584 GMT [1] LOG:  received smart shutdown request
```

7. Lets port-forward the keycloak service to access the UI in a browser

```
kubectl get svc -n keycloak
kubectl port-forward svc/<keycloak-service> -n keycloak <port-to-expose>/80
```

8. Navigate to a browser and access `localhost:<port-to-expose>`



9. Go to Administration Console - it will prompt you for a username and password. The default username is **user** and we set the new password to **Test**.
Create a Realm.

KEYCLOAK

user

master

master

Create Realm

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

1

Upload a JSON file

Realm name *

Enabled

☒ On

Create

Cancel

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

1

Upload a JSON file

Realm name *

new-realm

Enabled

☒ On

Create

Cancel

Realm created successfully



Welcome to new-realm

If you want to leave this page and manage this realm, please click the corresponding menu items in the left navigation bar.

10. Create a new user in the new realm

Users

Users are the users in the current realm. [Learn more](#)

User list

Default search

Search user



Add user

Delete user

0 - 0



No search results

Click on the search bar above to search

Create user

Required user actions ?

Username *

Email

Email verified ? ☐ No

First name

Last name

Groups ?

11. Update the new users password

[Users](#) > [User details](#)

test-user1

Enabled

Action ▾

Details

Attributes

Credentials

Role mapping

Groups

Consents

Identity provider links

Sessions

ID *

7b105b4d-97e1-4ea0-b825-1ee907bae92f

Created at *

12/7/2023, 11:52:59 AM

Required user actions

Select action ▾

Username *

test-user1

Email

Email verified ?

No

First name

Last name

Save

Revert

/users/7b105b4d-97e1-4ea0-b825-1ee907bae92f/settings

[Users](#) > [User details](#)

test-user1

Enabled

Action ▾

Details

Attributes

Credentials

Role mapping

Groups

Consents

Identity provider links

Sessions

No credentials

This user does not have any credentials. You can set password for this user.

Set password

Set password for test-user1

Password *

.....

👁

Password confirmation *

.....

👁

Temporary ?

On

Save

Cancel

Users > User details

test-user1

Enabled

Action

Details

Attributes

Credentials

Role mapping

Groups

Consents

Identity provider links

Sessions

?	Type	User label	Data
	Password	My password	<div>Show data</div> <div>Reset password</div> <div></div>

12. Create a new client - passing in rootURL. This port used here is the port where the application will be running.

The screenshot shows the Keycloak Admin Console interface for configuring a new client. On the left is a sidebar with navigation options: Manage, Clients (selected), Client scopes, Realm roles, Users, Groups, Sessions, Events, Configure, Realm settings, Authentication, Identity providers, and User federation. The main content area is titled 'node' with a sub-label 'OpenID Connect'. It includes a toggle switch for 'Enabled' (checked) and an 'Action' dropdown menu. Below this is a description: 'Clients are applications and services that can request authentication of a user.' There are five tabs: Settings (active), Roles, Client scopes, Sessions, and Advanced. The 'General Settings' section contains fields for 'Client ID' (node), 'Name' (test-client), and 'Description'. An 'Always display in UI' toggle is set to 'Off'. The 'Access settings' section includes 'Root URL' (http://localhost:3000), 'Home URL' (empty), and 'Valid redirect URIs' (http://localhost:3000/*) with a link to 'Add valid redirect URIs'. A 'Jump to section' sidebar on the right lists: General Settings, Access settings, Capability config, Login settings, and Logout settings. At the bottom are 'Save' and 'Revert' buttons.

13. Once created, you will need to export the json of the client from the Action tab > Export. This will download the client.json

This screenshot shows the same 'node' client configuration page, but with the 'Action' dropdown menu open. The menu options are: 'Download adapter config', 'Export' (highlighted), and 'Delete'. The 'General Settings' section is partially visible, showing the 'Client ID' field with the value 'node'. The 'Jump to section' sidebar on the right shows 'General Settings' as the selected option. A file download notification at the top right indicates 'node.json' (1,316 B) has been downloaded.

```

{
  "clientId": "node",
  "name": "test-client",
  "description": "",
  "rootUrl": "http://localhost:3000",
  "adminUrl": "http://localhost:3000",
  "baseUrl": "",
  "surrogateAuthRequired": false,
  "enabled": true,
  "alwaysDisplayInConsole": false,
  "clientAuthenticatorType": "client-secret",
  "redirectUris": [
    "http://localhost:3000/*"
  ],
  "webOrigins": [
    "http://localhost:3000"
  ],
  "notBefore": 0,
  "bearerOnly": false,
  "consentRequired": false,
  "standardFlowEnabled": true,
  "implicitFlowEnabled": false,
  "directAccessGrantsEnabled": true,
  "serviceAccountsEnabled": false,
  "publicClient": true,
  "frontchannelLogout": true,
  "protocol": "openid-connect",
  "attributes": {
    "oidc.ciba.grant.enabled": "false",
    "oauth2.device.authorization.grant.enabled": "false",
    "backchannel.logout.session.required": "true",
    "backchannel.logout.revoke.offline.tokens": "false"
  },
  "authenticationFlowBindingOverrides": {},
  "fullScopeAllowed": true,
  "nodeReRegistrationTimeout": -1,
  "defaultClientScopes": [
    "web-origins",
    "acr",
    "profile",
    "roles",
    "email"
  ],
  "optionalClientScopes": [
    "address",
    "phone",
    "offline_access",
    "microprofile-jwt"
  ],
  "access": {

```

14. You will then need to integrate this json with the application programming language that you choose. The steps to do this is simple to replicate and can vary. Keycloak comes with client adapters for most popular languages including Node.js, Python, Java etc.

Customise Helm Chart to Deploy Realm, Client and User with Client URL.

To avoid carrying out these manual steps, we can customise the helm values file to deploy keycloak with a new realm, users, roles and clients that will configure the rootUrl of the client

Note: The rootUrl and adminUrl need to be specified here with the Client URL

Update values.yaml with the json

```
keycloakConfigCli:
  enabled: true
  configuration:
    realm1.json: |
      {
        "realm": "realm-test1",
        "enabled": "true",
        "roles": {
          "realm": [
            {
              "name": "user",
              "composite": false,
              "clientRole": false
            },
            {
              "name": "admin",
              "composite": false,
              "clientRole": false
            }
          ]
        },
        "users": [
          {
            "username": "realm-test-user1",
            "enabled": true,
            "realmRoles": [
              "user"
            ],
            "credentials": [
              {
                "type": "password",
                "value": "test"
              }
            ]
          }
        ]
      },
    clients: [
      {
        "clientId": "client-test",
        "enabled": "true",
        "directAccessGrantsEnabled": true,
        "redirectUris": [""],
        "webOrigins": [""],
        "bearerOnly": false,
        "rootUrl": "http://localhost:3000",
        "adminUrl": "http://localhost:3000"
      }
    ]
  }
```

Conclusion

The final values file will look like this:

```

global:
  storageClass: gp2
tls:
  enabled: true
  autoGenerated: true
auth:
  adminPassword: Test
postgresql:
  auth:
    password: test
    postgresPassword: test
keycloakConfigCli:
  enabled: true
  configuration:
    realm1.json: |
      {
        "realm": "realm-test1",
        "enabled": "true",
        "roles": {
          "realm": [
            {
              "name": "user",
              "composite": false,
              "clientRole": false
            },
            {
              "name": "admin",
              "composite": false,
              "clientRole": false
            }
          ]
        },
        "users": [
          {
            "username": "realm-test-user1",
            "enabled": true,
            "realmRoles": [
              "user"
            ],
            "credentials": [
              {
                "type": "password",
                "value": "test"
              }
            ]
          }
        ]
      },
    clients: [
      {
        "clientId": "client-test",
        "enabled": "true",
        "directAccessGrantsEnabled": true,
        "redirectUris": [""],
        "webOrigins": [""],
        "bearerOnly": false,
        "rootUrl": "http://localhost:3000",
        "adminUrl": "http://localhost:3000"
      }
    ]
  }
}

```