Red Hat build of Keycloak 24.0

Operator ガイド

Last Updated: 2024-06-06

法律上の通知

概要

このガイドには、Red Hat build of Keycloak 24.0 Operator の設定と使用に関する管理者向け情報が記載されています。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 RED HAT BUILD OF KEYCLOAK OPERATOR のインストール

この手順を使用して、Red Hat build of Keycloak Operator を OpenShift クラスターにインストールします。

1. OpenShift Container Platform Web コンソールを開きます。
2. 左側の列で、**Home**、**Operators**、**OperatorHub** をクリックします。
3. 検索入力ボックスで "Keycloak" を検索します。
4. 結果のリストから Operator を選択します。
5. 画面の指示に従います。

CLI または Web コンソールを使用して Operator をインストールする一般的な手順については、Installing Operators in your namespace を参照してください。デフォルトのカタログでは、Operator の名前は **rhbk-operator** です。目的の Red Hat build of Keycloak バージョンに対応するチャネルを必ず使用してください。

第2章 基本的な RED HAT BUILD OF KEYCLOAK デプロイメント

2.1. 基本的な RED HAT BUILD OF KEYCLOAK デプロイメントの実行

この章では、Operator を使用して OpenShift 上で基本的な Red Hat build of Keycloak デプロイメントを実行する方法について説明します。

2.1.1. デプロイメントの準備

Red Hat build of Keycloak Operator がインストールされ、クラスター namespace で実行されたら、他のデプロイメントの要件をセットアップできます。

- データベース
- ホスト名
- TLS 証明書と関連する鍵

2.1.1.1. データベース

データベースが利用可能であり、Red Hat build of Keycloak がインストールされているクラスター namespace からアクセスできる必要があります。サポートされているデータベースのリストについては、データベースの設定 を参照してください。Red Hat build of Keycloak Operator はデータベースを管理しないため、管理者がプロビジョニングする必要があります。クラウドプロバイダーのサービスを確認するか、データベース Operator の使用を検討してください。

開発目的には、一時的な PostgreSQL Pod インストールを使用できます。プロビジョニングするには、以下の方法に従います。

YAML ファイル **example-postgres.yaml** を作成します。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgresql-db
spec:
  serviceName: postgresql-db-service
  selector:
    matchLabels:
      app: postgresql-db
  replicas: 1
  template:
    metadata:
      labels:
        app: postgresql-db
    spec:
      containers:
        - name: postgresql-db
          image: postgres:15
          volumeMounts:
            - mountPath: /data
              name: cache-volume
          env:
            - name: POSTGRES_USER
```

```
            value: testuser
          - name: POSTGRES_PASSWORD
            value: testpassword
          - name: PGDATA
            value: /data/pgdata
          - name: POSTGRES_DB
            value: keycloak
      volumes:
        - name: cache-volume
          emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-db
spec:
  selector:
    app: postgresql-db
  type: LoadBalancer
  ports:
  - port: 5432
    targetPort: 5432
```

変更を適用します。

```
oc apply -f example-postgres.yaml
```

2.1.1.2. ホスト名

実稼働環境に対応したインストールの場合、Red Hat build of Keycloak に接続するために使用できるホスト名が必要です。利用可能な設定については、ホスト名の設定 を参照してください。

この章では開発目的で **test.keycloak.org** を使用します。

OpenShift 上で実行する場合は、Ingress が有効で、spec.ingress.classname が openshift-default に設定されている場合、Keycloak CR の spec.hostname.hostname を未設定のままにしておくことができます。Operator は、明示的なホストのない OpenShift ルートによって作成されるホスト名と同様に、保存されたバージョンの CR にデフォルトのホスト名 ingress-namespace.appsDomain を割り当てます。appsDomain が変更された場合、または何らかの理由で別のホスト名が必要な場合は、Keycloak CR を更新してください。

2.1.1.3. TLS 証明書と鍵

証明書と鍵を取得するには、認証局に問い合わせてください。

開発目的の場合は、次のコマンドを入力して自己署名証明書を取得できます。

```
openssl req -subj '/CN=test.keycloak.org/O=Test Keycloak./C=US' -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 365 -out certificate.pem
```

次のコマンドを入力して、クラスター namespace に証明書をシークレットとしてインストールする必要があります。

```
oc create secret tls example-tls-secret --cert certificate.pem --key key.pem
```

2.1.2. Red Hat build of Keycloak のデプロイ

Red Hat build of Keycloak をデプロイするには、Keycloak カスタムリソース定義 (CRD) に基づいてカスタムリソース (CR) を作成します。

データベース認証情報を別のシークレットに保存することを検討してください。次のコマンドを入力します。

```
oc create secret generic keycloak-db-secret \
  --from-literal=username=[your_database_username] \
  --from-literal=password=[your_database_password]
```

Keycloak CRD を使用して、いくつかのフィールドをカスタマイズできます。基本的なデプロイメントの場合は、次のアプローチに従うことができます。

YAML ファイル **example-kc.yaml** を作成します。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  instances: 1
  db:
    vendor: postgres
    host: postgres-db
    usernameSecret:
      name: keycloak-db-secret
      key: username
    passwordSecret:
      name: keycloak-db-secret
      key: password
  http:
    tlsSecret: example-tls-secret
  hostname:
    hostname: test.keycloak.org
  proxy:
    headers: xforwarded # double check your reverse proxy sets and overwrites the X-Forwarded-* headers
```

変更を適用します。

```
oc apply -f example-kc.yaml
```

Red Hat build of Keycloak インスタンスがクラスターにプロビジョニングされていることを確認するには、次のコマンドを入力して、作成された CR のステータスを確認します。

```
oc get keycloaks/example-kc -o go-template='{{range .status.conditions}}CONDITION: {{.type}}{{"\n"}}  STATUS: {{.status}}{{"\n"}}  MESSAGE: {{.message}}{{"\n"}}{{end}}'
```

デプロイメントの準備ができたら、次のような出力を探します。

```
CONDITION: Ready
  STATUS: true
```

```
  MESSAGE:
CONDITION: HasErrors
  STATUS: false
  MESSAGE:
CONDITION: RollingUpdate
  STATUS: false
  MESSAGE:
```

2.1.3. Red Hat build of Keycloak デプロイメントへのアクセス

Red Hat build of Keycloak デプロイメントは、基本的な Ingress を通じて公開され、指定されたホスト名を通じてアクセス可能になります。複数のデフォルトの IngressClass インスタンスを含むインストールの場合、または OpenShift 4.12 以降で実行する場合は、**className** プロパティーを持つ **ingress** 仕様を目的のクラス名に設定して、ingressClassName を指定する必要があります。

YAML ファイル **example-kc.yaml** を編集します。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
    ...
    ingress:
      className: openshift-default
```

デフォルトの Ingress がユースケースに適合しない場合は、**enabled** プロパティーを持つ **ingress** 仕様を **false** 値に設定して無効にします。

YAML ファイル **example-kc.yaml** を編集します。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
    ...
    ingress:
      enabled: false
```

変更を適用します。

```
oc apply -f example-kc.yaml
```

サービス **<keycloak-cr-name>-service** を参照する代替 Ingress リソースを提供できます。

デバッグと開発の目的では、ポート転送を使用して Red Hat build of Keycloak サービスに直接接続することを検討してください。たとえば、次のコマンドを入力します。

```
oc port-forward service/example-kc-service 8443:8443
```

2.1.3.1. Ingress コントローラーに一致するリバースプロキシー設定の指定

Operator は、**Forwarded** ヘッダーや **X-Forwarded-*** ヘッダーなど、サーバーが受け入れるリバースプロキシーヘッダーの設定をサポートしています。

Ingress 実装で **Forwarded** または **X-Forwarded-*** ヘッダーのいずれかを設定して上書きする場合は、次のようにすることで、それを Keycloak CR に反映できます。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ...
  proxy:
    headers: forwarded|xforwarded
```

注記

proxy.headers フィールドが指定されていない場合、Operator はデフォルトで **proxy=passthrough** を暗黙的に設定して、従来の動作にフォールバックします。これにより、サーバーログに非推奨の警告が記録されます。このフォールバックは今後のリリースで削除される予定です。

警告

proxy.headers フィールドを使用する場合は、Ingress によって **Forwarded** ヘッダーまたは **X-Forwarded-*** ヘッダーが適切に設定および上書きされることを確認してください。これらのヘッダーを設定するには、Ingress コントローラーのドキュメントを参照してください。パススルー TLS は Ingress によるリクエストヘッダーの変更を許可しないため、再暗号化またはエッジ TLS 終端用にコントローラーを設定することを検討してください。設定を誤ると、Red Hat build of Keycloak がセキュリティー上の脆弱性にさらされることになります。

詳細は、リバースプロキシーの使用 ガイドを参照してください。

2.1.4. 管理コンソールへのアクセス

Red Hat build of Keycloak をデプロイする場合、Operator は任意の初期管理者の **username** と **password** を生成し、それらの認証情報を CR と同じ namespace に basic-auth シークレットオブジェクトとして保存します。

警告

実稼働を開始する前に、デフォルトの管理者の認証情報を変更し、Red Hat build of Keycloak で MFA を有効にしてください。

初期の管理者認証情報を取得するには、シークレットを読み取ってデコードする必要があります。シークレット名は、Keycloak CR 名に固定接尾辞 **-initial-admin** を加えたものから導出されます。**example-kc** CR のユーザー名とパスワードを取得するには、次のコマンドを入力します。

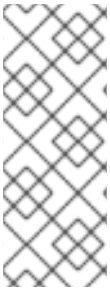
```
oc get secret example-kc-initial-admin -o jsonpath='{.data.username}' | base64 --decode
oc get secret example-kc-initial-admin -o jsonpath='{.data.password}' | base64 --decode
```

これらの認証情報を使用して、管理コンソールまたは管理 REST API にアクセスできます。

第3章 RED HAT BUILD OF KEYCLOAK レルムのインポート

3.1. RED HAT BUILD OF KEYCLOAK レルムのインポート

Red Hat build of Keycloak Operator を使用すると、Keycloak デプロイメントのレルムのインポートを実行できます。



注記

- Red Hat build of Keycloak に同じ名前のレルムがすでに存在する場合、上書きされません。
- Realm Import CR は、新しいレルムの作成のみをサポートし、それらの更新や削除は行いません。Red Hat build of Keycloak で直接実行されたレルムへの変更は、CR に同期されません。

3.1.1. レルムインポートカスタムリソースの作成

以下は、レルムインポートカスタムリソース (CR) の例です。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: KeycloakRealmImport
metadata:
  name: my-realm-kc
spec:
  keycloakCRName: <name of the keycloak CR>
  realm:
    ...
```

この CR は、フィールド **keycloakCRName** で定義された Keycloak デプロイメント CR と同じ namespace に作成する必要があります。**realm** フィールドは完全な RealmRepresentation を受け入れます。

RealmRepresentation を取得する推奨方法は、エクスポート機能 レルムのインポートとエクスポート を利用することです。

1. レルムを単一のファイルにエクスポートします。
2. JSON ファイルを YAML に変換します。
3. 取得した YAML ファイルをコピーして **realm** キーのボディーとして貼り付け、インデントが正しいことを確認します。

3.1.2. Realm Import CR の適用

oc を使用して、正しいクラスター namespace に CR を作成します。

YAML ファイル **example-realm-import.yaml** を作成します。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: KeycloakRealmImport
metadata:
  name: my-realm-kc
spec:
```

```
  keycloakCRName: <name of the keycloak CR>
  realm:
    id: example-realm
    realm: example-realm
    displayName: ExampleRealm
    enabled: true
```

変更を適用します。

```
oc apply -f example-realm-import.yaml
```

実行中のインポートのステータスを確認するには、次のコマンドを入力します。

```
oc get keycloakrealmimports/my-realm-kc -o go-template='{{range .status.conditions}}CONDITION:
{{.type}}{{"\n"}}  STATUS: {{.status}}{{"\n"}}  MESSAGE: {{.message}}{{"\n"}}{{end}}'
```

インポートが正常に完了すると、出力は次の例のようになります。

```
CONDITION: Done
  STATUS: true
  MESSAGE:
CONDITION: Started
  STATUS: false
  MESSAGE:
CONDITION: HasErrors
  STATUS: false
  MESSAGE:
```

第4章 詳細設定

4.1. 詳細設定

この章では、Red Hat build of Keycloak デプロイメントの高度な設定にカスタムリソース (CR) を使用する方法について説明します。

4.1.1. サーバー設定の詳細

多くのサーバーオプションは、Keycloak CR のファーストクラスシチズンフィールドとして公開されます。CR の構造は、Red Hat build of Keycloak の設定構造に基づいています。たとえば、サーバーの **https-port** を設定するには、CR で同様のパターンに従い、**httpsPort** フィールドを使用します。次の例は、複雑なサーバー設定です。ただし、サーバーオプションと Keycloak CR の関係を示しています。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  db:
    vendor: postgres
    usernameSecret:
      name: usernameSecret
      key: usernameSecretKey
    passwordSecret:
      name: passwordSecret
      key: passwordSecretKey
    host: host
    database: database
    port: 123
    schema: schema
    poolInitialSize: 1
    poolMinSize: 2
    poolMaxSize: 3
  http:
    httpEnabled: true
    httpPort: 8180
    httpsPort: 8543
    tlsSecret: my-tls-secret
  hostname:
    hostname: my-hostname
    admin: my-admin-hostname
    strict: false
    strictBackchannel: false
  features:
    enabled:
      - docker
      - authorization
    disabled:
      - admin
      - step-up-authentication
  transaction:
    xaEnabled: false
```

オプションのリストについては、Keycloak CRD を参照してください。オプションの設定の詳細は、すべての設定 を参照してください。

4.1.1.1. 追加オプション

一部のエキスパートサーバーオプションは、Keycloak CR の専用フィールドとして使用できません。除外されているフィールドには、たとえば次のものがあります。

- 基礎となる Red Hat build of Keycloak 実装についての深い理解を必要とするフィールド
- OpenShift 環境に関係のないフィールド
- プロバイダー設定用のフィールド (使用されるプロバイダーの実装に応じて動的であるため)

Keycloak CR の **additionalOptions** フィールドを使用すると、Red Hat build of Keycloak がキーと値のペアの形式で利用可能な設定を受け入れることができます。このフィールドを使用して、Keycloak CR で除外されているオプションを含めることができます。オプションの設定の詳細は、すべての設定 を参照してください。

この例に示すように、値はプレーンテキスト文字列またはシークレットオブジェクト参照として表現できます。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ...
  additionalOptions:
    - name: spi-connections-http-client-default-connection-pool-size
      secret: # Secret reference
        name: http-client-secret # name of the Secret
        key: poolSize # name of the Key in the Secret
    - name: spi-email-template-mycustomprovider-enabled
      value: true # plain text value
```

注記

この方法で定義されたオプションの名前形式は、設定ファイルで指定されたオプションのキー形式と同じです。さまざまな設定形式の詳細は、Red Hat build of Keycloak の設定 を参照してください。

4.1.2. シークレット参照

シークレット参照は、Keycloak CR の一部の専用オプション (**tlsSecret** など) によって、または **additionalOptions** の値として使用されます。

同様に、ConfigMap 参照も **configMapFile** などのオプションによって使用されます。

シークレット参照や ConfigMap 参照を指定する場合は、参照されるキーを含むシークレットまたは ConfigMap が、それを参照する CR と同じ namespace に存在することを確認してください。

Operator は、参照されるシークレットまたは ConfigMaps の変更を約 1 分ごとにポーリングします。有効な変更が検出されると、Operator は Red Hat build of Keycloak デプロイメントのローリング再起動を実行して、変更を取得します。

4.1.3. サポートされない機能

CR の **unsupported** フィールドには、完全にはテストされておらず、テクノロジープレビューである非常に実験的な設定オプションが含まれています。

4.1.3.1. Pod テンプレート

Pod テンプレートは、デプロイメントテンプレートに使用される raw API 表現です。このフィールドは、使用例の CR の最上位にサポートされているフィールドが存在しない場合の一時的な回避策です。

Operator は、提供されたテンプレートのフィールドを、特定のデプロイメント用に Operator が生成した値とマージします。この機能を使用すると、高度なカスタマイズにアクセスできます。ただし、デプロイメントが期待どおりに機能するという保証はありません。

次の例は、ラベル、アノテーション、ボリューム、およびボリュームマウントの挿入を示しています。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ...
  unsupported:
    podTemplate:
      metadata:
        labels:
          my-label: "keycloak"
      spec:
        containers:
          - volumeMounts:
              - name: test-volume
                mountPath: /mnt/test
        volumes:
          - name: test-volume
            secret:
              secretName: keycloak-additional-secret
```

4.1.4. 必須オプションの無効化

Red Hat build of Keycloak と Red Hat build of Keycloak Operator は、セキュリティーを考慮した、実稼働環境に対応した最適なエクスペリエンスを提供します。ただし、開発段階では、主要なセキュリティー機能を無効にすることができます。

具体的には、次の例に示すように、ホスト名と TLS を無効にできます。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ...
  http:
    httpEnabled: true
```

```
  hostname:
    strict: false
    strictBackchannel: false
```

4.1.5. リソース要件

Keycloak CR では、Red Hat build of Keycloak コンテナーのコンピュートリソースを管理するための **resources** オプションを指定できます。これにより、Keycloak CR を使用してメインの Keycloak デプロイメントに対して、および Realm Import CR を使用してレルムインポートジョブに対して、個別にリソースを要求および制限できます。

値が指定されていない場合、デフォルトの **requests** メモリーが **1700MiB** に設定され、**limits** メモリーが **2GiB** に設定されます。これらの値は、Red Hat build of Keycloak メモリー管理の詳細な分析に基づいて選択されています。

Realm Import CR に値が指定されていない場合は、Keycloak CR に指定されている値、または上記のデフォルトにフォールバックします。

次のように、要件に応じてカスタム値を指定できます。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ...
  resources:
    requests:
      cpu: 1200m
      memory: 896Mi
    limits:
      cpu: 6
      memory: 3Gi
```

さらに、Red Hat build of Keycloak コンテナーでは、ヒープサイズの相対値を指定して、ヒープサイズをより効率的に管理できます。これは、特定の JVM オプションを指定することで実現できます。

詳細は、Red Hat build of Keycloak をコンテナー内で実行する を参照してください。

4.1.6. トラストストア

信頼済み証明書を提供する必要がある場合、信頼済み証明書の設定 で説明されているように、Keycloak CR で、サーバーのトラストストアを設定するための最上位レベルの機能を使用できます。

Keycloak spec の truststores フィールドを使用して、PEM でエンコードされたファイル、または拡張子が **.p12** または **.pfx** の PKCS12 ファイルを含むシークレットを指定します。以下に例を示します。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ...
  truststores:
```

```
    my-truststore:
      secret:
        name: my-secret
```

my-secret の内容が PEM ファイルである場合の例を以下に示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
stringData:
  cert.pem: |
    -----BEGIN CERTIFICATE-----
    ...
```

Kubernetes または OpenShift 環境で実行する場合、信頼できる証明書の既知の場所が自動的に追加されます。これには、/var/run/secrets/kubernetes.io/serviceaccount/ca.crt と、/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt (存在する場合) が含まれます。

第5章 カスタム RED HAT BUILD OF KEYCLOAK イメージの使用

5.1. OPERATOR を使用した RED HAT BUILD OF KEYCLOAK カスタムイメージ

Keycloak カスタムリソース (CR) を使用すると、Red Hat build of Keycloak サーバーのカスタムコンテナーイメージを指定できます。

注記

Operator とオペランドの完全な互換性を確保するには、カスタムイメージで使用される Red Hat build of Keycloak リリースのバージョンが、Operator のバージョンと一致していることを確認してください。

5.1.1. ベストプラクティス

デフォルトの Red Hat build of Keycloak イメージを使用する場合、サーバーは Pod が起動するたびにコストのかかる再オーグメンテーションを実行します。この遅延を回避するには、イメージのビルド時に、オーグメンテーションを組み込んだカスタムイメージを提供します。

カスタムイメージを使用すると、コンテナーのビルド中に Keycloak の **ビルド時** 設定と機能拡張を指定することもできます。

このようなイメージをビルドする方法については、コンテナーでの Red Hat build of Keycloak の実行 を参照してください。

5.1.2. カスタム Red Hat build of Keycloak イメージの提供

カスタムイメージを提供するには、次の例に示すように Keycloak CR で **image** フィールドを定義します。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  instances: 1
  image: quay.io/my-company/my-keycloak:latest
  http:
    tlsSecret: example-tls-secret
  hostname:
    hostname: test.keycloak.org
```

注記

カスタムイメージを使用すると、すべてのビルド時オプションが専用フィールドを介して渡されるか、**additionalOptions** が無視されます。

5.1.3. 最適化されていないカスタムイメージ

事前に拡張されたイメージを使用することが推奨されますが、最適化されていないカスタムイメージや、拡張されたイメージでビルド時プロパティーを使用することも可能です。次の例に示すように、**startOptimzed** フィールドを **false** に設定するだけです。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  instances: 1
  image: quay.io/my-company/my-keycloak:latest
  startOptimized: false
  http:
    tlsSecret: example-tls-secret
  hostname:
    hostname: test.keycloak.org
```

起動するたびに再オーグメンテーションコストが発生することに注意してください。