Red Hat build of Keycloak 24.0

Operator 指南

Last Updated: 2024-06-28

法律通告

摘要

本指南包含管理员用来配置和使用红帽构建的 Keycloak 24.0 Operator 的信息。

目录

第 1 章 RED HAT BUILD OF KEYCLOAK OPERATOR 安装

使用这个流程在 OpenShift 集群中安装 Keycloak Operator 的红帽构建。

1. 打开 OpenShift Container Platform Web 控制台。
2. 在左列中，点 **Home**,**Operators**,**OperatorHub**。
3. 在搜索输入框中搜索 "Keycloak"。
4. 从结果列表中选择 Operator。
5. 按照屏幕上的说明进行操作。

有关使用 CLI 或 Web 控制台安装 Operator 的常规说明，请参阅 在命名空间中安装 Operator。在默认目录中，Operator 名为 **rhbk-operator**。确保使用与所需红帽构建的 Keycloak 版本对应的频道。

第 2 章 基本红帽构建的 KEYCLOAK 部署

2.1. 执行基本红帽构建的 KEYCLOAK 部署

本章论述了如何使用 Operator 在 OpenShift 中执行基本红帽 Keycloak 部署构建。

2.1.1. 准备部署

在 Red Hat build of Keycloak Operator 安装并在集群命名空间中运行后，您可以设置其他部署先决条件。

- 数据库
- 主机名
- TLS 证书和相关密钥

2.1.1.1. 数据库

数据库应该可从安装红帽构建的 Keycloak 的集群命名空间访问。有关支持的数据库列表，请参阅配置数据库。红帽构建的 Keycloak Operator 不管理数据库，您需要自行置备。考虑验证您的云供应商产品或使用数据库操作器。

出于开发目的，您可以使用临时 PostgreSQL pod 安装。要置备它，请遵循以下方法：

创建 YAML 文件 **example-postgres.yaml** ：

```
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
```

应用更改：

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

2.1.1.2. 主机名

对于生产环境就绪安装，您需要一个可用于联系红帽构建的 Keycloak 的主机名。请参阅为可用配置配置主机名。https://access.redhat.com/documentation/zh-cn/red_hat_build_of_keycloak/24.0/html-single/server_guide/#hostname-

出于开发目的，本章将使用 **test.keycloak.org**。

在 OpenShift 上运行时，启用了 ingress，并将 spec.ingress.classname 设置为 openshift-default，您可能会在 Keycloak CR 中保留 spec.hostname.hostname。Operator 将为 CR 的存储版本分配一个默认主机名，类似于没有显式主机的 OpenShift Route 创建的内容 - 为 ingress-namespace.appsDomain 如果 appsDomain 更改，或者您需要不同的主机名来更新 Keycloak CR。

2.1.1.3. TLS 证书和密钥

查看您的认证机构以获取证书和密钥。

出于开发目的，您可以输入以下命令获取自签名证书：

```
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
```

您应该通过输入以下命令将它们安装到集群命名空间中作为 Secret：

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

2.1.2. 部署红帽构建的 Keycloak

要部署红帽构建的 Keycloak，您可以根据 Keycloak 自定义资源定义(CRD)创建自定义资源(CR)。

考虑将数据库凭证存储在单独的 Secret 中。输入以下命令：

-

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

您可以使用 Keycloak CRD 自定义几个字段。对于基本部署，您可以遵循以下方法：

创建 YAML 文件 **example-kc.yaml** ：

```
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
```

应用更改：

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

要检查红帽构建的 Keycloak 实例是否已在集群中置备，请输入以下命令检查创建的 CR 的状态：

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

部署就绪后，查找类似如下的输出：

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

2.1.3. 访问红帽构建的 Keycloak 部署

红帽构建的 Keycloak 部署通过基本 Ingress 公开，并可通过提供的主机名访问。在具有多个默认 IngressClass 实例的安装中，或者在 OpenShift 4.12+ 上运行时，您应该通过将 **ingress** spec 和 **className** 属性设置为所需类名称来提供 ingressClassName ：

编辑 YAML 文件 **example-kc.yaml** ：

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

如果默认入口不适合您的用例，请通过将 **ingress** spec 设置为 **false** 值来禁用它：

编辑 YAML 文件 example-kc.yaml ：

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

应用更改：

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

您可以提供指向服务 < keycloak-cr-name>-service 的替代入口资源。

为了调试和开发目的，请考虑使用端口转发直接连接到红帽构建的 Keycloak 服务。例如，输入以下命令：

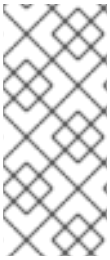
```
oc port-forward service/example-kc-service 8443:8443
```

2.1.3.1. 配置与 Ingress Controller 匹配的反向代理设置

Operator 支持配置服务器应接受哪些反向代理标头，其中包括 Forwarded 和 X-Forwarded34） 标头。

如果 Ingress 实现设置并覆盖 Forwarded 或 X-Forwarded114 标头，您可以在 Keycloak CR 中反映它，如下所示：

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



注意

如果没有指定 proxy.headers 字段，Operator 会默认隐式设置 proxy=passthrough 来回退到传统的行为。这会在服务器日志中产生弃用警告。此回退将在以后的发行版本中被删除。

警告

使用 proxy.headers 字段时，请确保您的 Ingress 正确设置并分别覆盖 Forwarded 或 X-Forwarded34） 标头。要设置这些标头，请参阅 Ingress Controller 的文档。考虑将其配置为重新加密或边缘 TLS 终止，因为 passthrough TLS 不允许 Ingress 修改请求标头。错误配置会将红帽构建的 Keycloak 暴露给安全漏洞。

详情请参考 使用反向代理 指南。

2.1.4. 访问管理控制台

在部署红帽构建的 Keycloak 时，Operator 会生成一个任意初始 admin 用户名和密码，并将这些凭证保存为与 CR 相同的命名空间中的 basic-auth Secret 对象。

警告

在进入生产环境前，更改默认的 admin 凭证并在红帽构建的 Keycloak 中启用 MFA。

要获取初始 admin 凭证，您必须读取和解码 Secret。Secret 名称来源于 Keycloak CR 名称，加上固定后缀 -initial-admin。要获取 example-kc CR 的用户名和密码，请输入以下命令：

```
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
```

您可以使用这些凭据来访问管理控制台或 Admin REST API。

第 3 章 红帽构建的 KEYCLOAK REALM IMPORT

3.1. 导入红帽构建的 KEYCLOAK REALM

使用红帽构建的 Keycloak Operator，您可以为 Keycloak Deployment 执行域导入。

注意

- 如果红帽构建的 Keycloak 中已存在具有相同名称的 Realm，则不会覆盖它。
- Realm Import CR 仅支持创建新域，且不更新或删除它们。直接在红帽构建的 Keycloak 上执行的域更改不会在 CR 中同步。

3.1.1. 创建 Realm Import 自定义资源

以下是 Realm Import 自定义资源(CR)的示例：

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

此 CR 应该与 Keycloak Deployment CR 在同一命名空间中创建，在字段 keycloakCRName 中定义。realm 字段接受完整的 RealmRepresentation。

获取 RealmRepresentation 的建议方法是利用导出功能 导入和导出域。

1. 将 Realm 导出至单个文件。
2. 将 JSON 文件转换为 YAML。
3. 复制并粘贴获取的 YAML 文件，作为 realm 键的正文，确保缩进正确。

3.1.2. 应用 Realm Import CR

使用 oc 在正确的集群命名空间中创建 CR：

创建 YAML 文件 example-realm-import.yaml ：

```
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
```

应用更改：

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

要检查正在运行的导入的状态，请输入以下命令：

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

当导入成功完成时，输出将类似以下示例：

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

第 4 章 高级配置

4.1. 高级配置

本章论述了如何使用自定义资源(CR)进行红帽构建的 Keycloak 部署高级配置。

4.1.1. 服务器配置详情

许多服务器选项都作为 Keycloak CR 中的第一类统计字段公开。CR 的结构基于红帽构建的 Keycloak 的配置结构。例如，若要配置服务器的 https-port，请按照 CR 中的类似模式并使用 httpsPort 字段。以下示例是复杂的服务器配置，但演示了服务器选项和 Keycloak CR 之间的关系：

```
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
```

有关选项列表，请参阅 Keycloak CRD。有关配置选项的详情，请查看 所有配置。

4.1.1.1. 其他选项

有些专家服务器选项作为 Keycloak CR 中的专用字段不可用。以下是省略的字段示例：

- 需要深入了解底层红帽构建的 Keycloak 实现的字段
- 与 OpenShift 环境无关的字段
- 供应商配置的字段，因为它们是根据使用的供应商实施动态的

Keycloak CR 的 additionalOptions 字段可让红帽构建 Keycloak 以键值对的形式接受任何可用的配置。您可以使用此字段包含 Keycloak CR 中省略的任何选项。有关配置选项的详情，请查看 所有配置。

这些值可以用纯文本字符串或 Secret 对象引用表示，如下例所示：

```
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
```

注意

以这种方式定义的选项名称格式与配置文件中指定的选项的密钥格式相同。有关各种配置格式的详情，请参阅配置红帽构建的 Keycloak。

4.1.2. Secret References

secret References 由 Keycloak CR 中的一些专用选项使用，如 tlsSecret，或作为 additionalOptions 中的值。

与 ConfigMap 引用类似，选项（如 configMapFile ）使用。

在指定 Secret 或 ConfigMap 参考时，请确保包含引用密钥的 Secret 或 ConfigMap 存在于与引用它的 CR 相同的命名空间中。

Operator 大约会每分钟轮询到引用 Secret 或 ConfigMap 的更改。当检测到有意义的更改时，Operator 会执行红帽构建的 Keycloak Deployment 的滚动重启，以获取更改。

4.1.3. 不支持的功能

CR 的 unsupported 字段包含没有完全测试且技术预览的高实验性配置选项。

4.1.3.1. Pod 模板

Pod 模板是用于 Deployment 模板的原始 API 表示。当您的用例的 CR 顶级不存在支持字段时，此字段是一个临时临时解决方案。

Operator 将提供的模板字段与 Operator 为特定 Deployment 生成的值合并。使用此功能，您可以访问高级别的自定义。但是，不能保证部署可以按预期工作。

以下示例演示了注入标签、注解、卷和卷挂载：

```
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. 禁用所需选项

红帽构建的 Keycloak 和红帽构建的 Keycloak Operator 提供了最佳生产就绪体验。但是，在开发阶段，您可以禁用关键安全功能。

具体来说，您可以禁用主机名和 TLS，如下例所示：

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

4.1.5. 资源要求

Keycloak CR 允许指定管理红帽构建的 Keycloak 容器 计算资源的 资源选项。它提供通过 Keycloak CR 独立请求和限制资源的功能，以及通过 Realm Import CR 的域导入作业。

如果没有指定值，则默认 请求内存 设置为 1700MiB，限值 内存设置为 2GiB。这些值根据红帽构建的 Keycloak 内存管理的更深入分析来选择。

如果没有在 Realm Import CR 中指定值，它将回退到 Keycloak CR 中指定的值，或回退到上面定义的默认值。

您可以根据您的要求指定自定义值，如下所示：

```
apiVersion: k8s.keycloak.org/v2alpha1
```

```
kind: Keycloak
metadata:
  name: example-kc
spec:
  ...
  resources:
    requests:
      cpu: 1200m
      memory: 896Mi
    limits:
      cpu: 6
      memory: 3Gi
```

此外，红帽构建的 Keycloak 容器通过提供堆大小的相对值来更有效地管理堆大小。它通过提供某些 JVM 选项来实现。

如需了解更多详细信息，请参阅 在容器中运行 Keycloak。

4.1.6. truststores

如果您需要提供可信证书，Keycloak CR 提供了一个顶级功能，用于配置服务器的信任存储，如 配置可信证书 中所述。

使用 Keycloak spec 的 truststores 小节来指定包含 PEM 编码文件的 Secret，或使用扩展名 .p12 或 .pfx 的 PKCS12 文件，例如：

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

其中 my-secret 的内容可以是 PEM 文件，例如：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
stringData:
```

```
  cert.pem: |
    -----BEGIN CERTIFICATE-----
    ...
```

在 Kubernetes 或 OpenShift 环境中自动包含可信证书已知位置时。这包括 /var/run/secrets/kubernetes.io/serviceaccount/ca.crt 和 /var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt （如果存在）。

第 5 章 使用自定义红帽构建的 KEYCLOAK 镜像

5.1. RED HAT BUILD OF KEYCLOAK CUSTOM IMAGE WITH THE OPERATOR

使用 Keycloak 自定义资源(CR)，您可以为红帽构建的 Keycloak 服务器指定自定义容器镜像。

注意

为确保 Operator 和 Operand 的完整兼容性，请确保自定义镜像中使用的红帽构建的 Keycloak 发行版本与 Operator 的版本一致。

5.1.1. 最佳实践

当使用默认红帽构建的 Keycloak 镜像时，服务器会在 Pod 启动时执行昂贵的重新错误。为了避免这种延迟，您可以为自定义镜像提供从构建镜像的构建时增加内置的自定义镜像。

使用自定义镜像，您还可以在容器构建期间指定 Keycloak *构建时* 配置和扩展。

有关如何构建此类镜像的说明，请参阅在容器中运行 Keycloak。

5.1.2. 提供自定义红帽构建的 Keycloak 镜像

要提供自定义镜像，您可以在 Keycloak CR 中定义 image 字段，如下例所示：

```
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
```

注意

使用自定义镜像时，每个构建时间选项都通过专用字段传递，或者忽略 additionalOptions。

5.1.3. 非优化的自定义镜像

虽然作为最佳实践方案，但如果您想使用非优化的自定义镜像或构建时间属性，则最好使用预装的镜像。您只需要将 startOptimzed 字段设置为 false，如下例所示：

```
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
```

请记住，这将在每次开始时造成重新增加成本。