Red Hat build of Quarkus 3.8

服务绑定

Last Updated: 2024-06-21

法律通告

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

探索服务绑定和工作负载投射，以了解它们需要连接到其他服务以获得其他信息检索。

目录

提供有关红帽构建的 QUARKUS 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

流程

1. 单击以下链接 **以创建 ticket**。
2. 在 **Summary** 中输入问题的简短描述。
3. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
4. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中有问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

第 1 章 服务绑定

本章提供有关添加到版本 2.7.5 的 Red Hat build of Quarkus 的服务绑定和工作负载投射信息，在版本 3.8 中 处于技术预览状态。

通常，OpenShift 应用程序和服务也称为可部署工作负载，需要连接到其他服务来检索其他信息，如服务 URL 或凭证。

Service Binding Operator 有助于检索必要信息，然后提供给应用程序和服务绑定工具（如 **quarkus-kubernetes-service-binding** 扩展），而无需直接影响或确定扩展工具本身的使用。

Quarkus 支持 Kubernetes 的服务绑定规格 将服务绑定到应用程序。

特别是，Quarkus 实现了规格 的工作负载投射 部分，使应用程序能够绑定到数据库或代理等服务，只需要最小配置。

要为可用扩展启用服务绑定，请将 **quarkus-kubernetes-service-binding** 扩展包含到应用程序依赖项。

- 您可以对服务绑定和工作负载投射使用以下扩展：
  - **quarkus-jdbc-mariadb**
  - **quarkus-jdbc-mssql**
  - **quarkus-jdbc-mysql**
  - **quarkus-jdbc-postgresql**
  - **Quarkus-mongo-client** - 技术预览
  - **quarkus-kafka-client**
  - **quarkus-smallrye-reactive-messaging-kafka**
  - **quarkus-reactive-mssql-client** - 技术预览
  - **quarkus-reactive-mysql-client**
  - **quarkus-reactive-pg-client**

1.1. 工作负载投射

工作负载投射是从 Kubernetes 集群获取服务配置的过程。此配置采用遵循某些约定的目录结构，并作为挂载的卷附加到应用程序或服务。

kubernetes-service-binding 扩展使用此目录结构来创建配置源，允许您配置其他模块，如数据库或消息代理。

您可以在应用程序开发期间使用工作负载投射将应用程序连接到开发数据库或其他本地运行的服务，而无需更改应用程序代码或配置。

有关在测试资源中包含目录结构并传递给集成测试的工作负载投射示例，请参阅 Kubernetes Service Binding datasource GitHub 仓库。

注意

- **k8s-sb** 目录是服务绑定的根目录。
  在本例中，只有一个名为 **fruit-db** 的数据库被绑定。此绑定数据库具有 **类型** 文件，该文件将 **postgresql** 指定为数据库类型，而 目录中的其他文件则提供必要的信息来建立连接。

- 当 Red Hat build of Quarkus 项目从 OpenShift Container Platform 设定的 **SERVICE_BINDING_ROOT** 环境变量中获取信息时，您可以找到文件系统中存在的生成的配置文件，并使用它们将 config-file 值映射到特定扩展的属性。

1.2. SERVICE BINDING OPERATOR 简介

Service Binding Operator 是一个 Operator，它为 Kubernetes 实现 Service Binding 规格，用于简化服务到应用程序的绑定。

支持 工作负载投射的容器化应用 以卷挂载的形式获取服务绑定信息。Service Binding Operator 读取绑定服务信息，并将其挂载到需要它的应用程序容器中。

应用程序和绑定服务之间的关联通过 **ServiceBinding** 资源表示，它声明了哪些服务要绑定到哪个应用程序。

Service Binding Operator 监视 **ServiceBinding** 资源，其告知 Operator 哪些应用程序要与哪些服务绑定。部署列出的应用程序时，Service Binding Operator 会收集必须传递给应用程序的所有绑定信息，然后通过使用绑定信息附加卷挂载来升级应用程序容器。

Service Binding Operator 完成以下操作：

- 观察绑定到特定服务的工作负载的 **ServiceBinding** 资源。
- 使用卷挂载将绑定信息应用到工作负载。

下面的章节描述了自动和半自动服务绑定方法及其用例。**kubernetes-service-binding** 扩展使用以下任一方法生成 **ServiceBinding** 资源。使用半自动化方法时，用户必须手动为目标服务提供配置。使用自动方法时，对于生成 **ServiceBinding** 资源的有限服务集合不需要额外的配置。

其他资源

- 工作负载投射

1.3. 半自动服务绑定

服务绑定过程从绑定到特定应用程序所需的服务的用户规格开始。这个表达式被 **kubernetes-service-binding** 扩展生成的 **ServiceBinding** 资源总结。使用 **kubernetes-service-binding** 扩展可帮助用户以最小配置生成 **ServiceBinding** 资源，从而简化进程整体。

然后，负责绑定进程的 Service Binding Operator 从 **ServiceBinding** 资源读取信息，并相应地将所需的文件挂载到容器中。

- **ServiceBinding** 资源的示例：

```
apiVersion: binding.operators.coreos.com/v1beta1
kind: ServiceBinding
metadata:
 name: binding-request
```

```
 namespace: service-binding-demo
spec:
 application:
   name: java-app
   group: apps
   version: v1
   resource: deployments
 services:
 - group: postgres-operator.crunchydata.com
   version: v1beta1
   kind: Database
   name: db-demo
   id: postgresDB
```

注意

- **quarkus-kubernetes-service-binding** 扩展提供了一种更紧凑的方式来表达相同的信息。例如：

```
quarkus.kubernetes-service-binding.services.db-demo.api-version=postgres-operator.crunchydata.com/v1beta1
quarkus.kubernetes-service-binding.services.db-demo.kind=Database
```

在 **application.properties** 中添加更早的配置属性后，**quarkus-kubernetes**，与 **quarkus-kubernetes-service-binding** 扩展相结合，会自动生成 **ServiceBinding** 资源。

之前提到的 **db-demo** 属性配置标识符现在有一个双角色，也完成以下操作：

- 将 **api-version** 和 **kind** 属性关联和组。
- 定义自定义资源的 **name** 属性，如果需要，您可以在以后编辑它。例如：

```
quarkus.kubernetes-service-binding.services.db-demo.api-version=postgres-operator.crunchydata.com/v1beta1
quarkus.kubernetes-service-binding.services.db-demo.kind=Database
quarkus.kubernetes-service-binding.services.db-demo.name=my-db
```

其他资源

- 如何将 Quarkus 与 Service Binding Operator 搭配使用
- 可绑定 Operator 列表

1.4. 使用分号(TRA-AUTOMATIC)方法生成 SERVICEBINDING 自定义资源

您可以以自动方式生成 **ServiceBinding** 资源。以下流程演示了 OpenShift Container Platform 部署过程，包括用于配置和部署应用程序的 Operator 的安装。

在此过程中，您可以从 Crunchy Data 安装 Service Binding Operator 和 PostgreSQL Operator 。

重要

PostgreSQL Operator 是一个第三方组件。对于 PostgreSQL Operator 支持策略和使用条款，请联系软件厂商 Crunchy Data。

然后，这个过程涉及创建 PostgreSQL 集群，设置一个简单的应用程序，然后部署并将其绑定到置备的集群。

先决条件

- 您已创建了 OpenShift Container Platform 4.12 集群。
- 具有 OperatorHub 和 OpenShift Container Platform 的管理员访问权限，以便从 OperatorHub 安装集群范围的 Operator。
- 已安装：
  - OpenShift、**oc**、编配工具
  - Maven 和 Java

流程

以下流程中的步骤使用 HOME (~)目录作为保存和安装目的地。

1. 使用 从 OpenShift Container Platform Web UI 安装 Service Binding Operator 过程安装 Service Binding Operator 版本 1.3.3 和更高版本。
   a. 验证安装：

   ```
   oc get csv -w
   ```

   当 Service Binding Operator 的 **阶段** 设置为 **Succeeded** 时，继续下一步。
2. 使用 Web 控制台或 CLI 从 OperatorHub 安装 Crunchy PostgreSQL Operator。
   a. 验证安装：

   ```
   oc get csv -w
   ```

   当 Operator 的 **阶段** 设置为 **Succeeded** 时，继续下一步。
3. 创建 PostgreSQL 集群：
   a. 创建新的 OpenShift Container Platform 命名空间，它将用于创建集群并在以后部署应用程序。此命名空间将在整个流程中称为 **demo**。

   ```
   oc new-project demo
   ```

   b. 创建以下自定义资源，并将它保存为 **pg-cluster.yml** ：

   ```
   apiVersion: postgres-operator.crunchydata.com/v1beta1
   kind: PostgresCluster
   metadata:
     name: hippo
   spec:
   ```

```
  openshift: true
  image: registry.developers.crunchydata.com/crunchydata/crunchy-postgres:ubi8-14.2-1
  postgresVersion: 14
  instances:
    - name: instance1
      dataVolumeClaimSpec:
        accessModes:
        - "ReadWriteOnce"
        resources:
          requests:
            storage: 1Gi
  backups:
    pgbackrest:
      image: registry.developers.crunchydata.com/crunchydata/crunchy-pgbackrest:ubi8-
2.38-0
      repos:
      - name: repo1
        volume:
          volumeClaimSpec:
            accessModes:
            - "ReadWriteOnce"
            resources:
              requests:
                storage: 1Gi
```

注意

此 YAML 已从 Service Binding Operator Quickstart 重复使用。

c. 应用创建的自定义资源：

```
oc apply -f ~/pg-cluster.yml
```

注意

此命令假设您将 **pg-cluster.yml** 文件保存到 HOME 目录中。

d. 检查 pod 以验证安装：

```
oc get pods -n demo
```

- 等待 Pod 进入 **READY** 状态，表示安装已完成。

4. 创建一个绑定到 PostgreSQL 数据库的 Quarkus 应用程序。
您要创建的应用程序是一个基本的 **todo** 应用程序，它使用 Hibernate 和 Panache 连接到 PostgreSQL。

a. 生成应用程序：

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:3.8.4.redhat-00002:create \
  -DplatformGroupId=com.redhat.quarkus.platform \
  -DplatformVersion=3.8.4.redhat-00002 \
  -DprojectGroupId=org.acme \
```

```
    -DprojectArtifactId=todo-example \
    -DclassName="org.acme.TodoResource" \
    -Dpath="/todo"
```

b. 添加连接到 PostgreSQL 所需的所有扩展，生成所有必需的资源，并为应用程序构建容器镜像：

```
./mvnw quarkus:add-extension -Dextensions="resteasy-reactive-jackson,jdbc-postgresql,hibernate-orm-panache,openshift,kubernetes-service-binding"
```

c. 如以下示例所示，创建一个简单的实体：

```
package org.acme;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;

import io.quarkus.hibernate.orm.panache.PanacheEntity;

@Entity
public class Todo extends PanacheEntity {

    @Column(length = 40, unique = true)
    public String title;

    public boolean completed;

    public Todo() {
    }

    public Todo(String title, Boolean completed) {
        this.title = title;
    }

}
```

d. 公开实体：

```
package org.acme;

import jakarta.transaction.Transactional;
import jakarta.ws.rs.*;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.core.Response.Status;
import java.util.List;

  @Path("/todo")
  public class TodoResource {

      @GET
      @Path("/")
      public List<Todo> getAll() {
        return Todo.listAll();
      }
```

```
    @GET
    @Path("/{id}")
    public Todo get(@PathParam("id") Long id) {
        Todo entity = Todo.findById(id);
        if (entity == null) {
            throw new WebApplicationException("Todo with id of " + id + " does not exist.",
Status.NOT_FOUND);
        }
        return entity;
    }

    @POST
    @Path("/")
    @Transactional
    public Response create(Todo item) {
        item.persist();
        return Response.status(Status.CREATED).entity(item).build();
    }

    @GET
    @Path("/{id}/complete")
    @Transactional
    public Response complete(@PathParam("id") Long id) {
        Todo entity = Todo.findById(id);
        entity.id = id;
        entity.completed = true;
        return Response.ok(entity).build();
    }


    @DELETE
    @Transactional
    @Path("/{id}")
    public Response delete(@PathParam("id") Long id) {
        Todo entity = Todo.findById(id);
        if (entity == null) {
            throw new WebApplicationException("Todo with id of " + id + " does not exist.",
Status.NOT_FOUND);
        }
        entity.delete();
        return Response.noContent().build();
    }

  }
```

5. 通过生成 **ServiceBinding** 资源来绑定到目标 PostgreSQL 集群。

   a. 提供服务协调来生成绑定并配置数据源：

   - apiVersion: **postgres-operator.crunchydata.com/v1beta1**
   - kind: **PostgresCluster**
   - 名称： **pg-cluster**
     这可以通过设置 **quarkus.kubernetes-service-binding.services.<id>.** 前缀来实现，如下例所示。**id** 用于将属性分组到一起，并可分配任何值。

```
quarkus.kubernetes-service-binding.services.my-db.api-version=postgres-operator.crunchydata.com/v1beta1
quarkus.kubernetes-service-binding.services.my-db.kind=PostgresCluster
quarkus.kubernetes-service-binding.services.my-db.name=hippo

quarkus.datasource.db-kind=postgresql
quarkus.hibernate-orm.database.generation=drop-and-create
quarkus.hibernate-orm.sql-load-script=import.sql
```

b. 创建一个带有一些初始数据的 **import.sql** 脚本：

```
INSERT INTO todo(id, title, completed) VALUES (nextval('hibernate_sequence'), 'Finish the blog post', false);
```

6. 部署应用程序，包括 **ServiceBinding** 并将其应用到集群：

```
mvn clean install -Dquarkus.kubernetes.deploy=true -DskipTests
```

等待部署完成。

验证

1. 验证部署：

```
oc get pods -n demo -w
```

2. 验证安装：

a. 端口转发至本地 HTTP 端口，然后访问 **/setuptools 端点**。

```
oc port-forward service/todo-example 8080:80
```

b. 在网页浏览器中打开以下 URL：

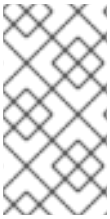
```
http://localhost:8080/todo
```

其他资源

- 如需更多信息，请参阅 快速入门指南 中的 Service Binding Operator 部分。

1.5. 自动服务绑定

当 **quarkus-kubernetes-service-binding** 扩展检测到需要访问兼容可绑定 Operator 提供的外部服务的应用程序时，它会自动生成 **ServiceBinding** 资源。



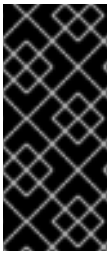
注意

只能为有限的服务类型生成自动服务绑定。

与既定的 Kubernetes 和 Quarkus 服务术语保持一致，本章使用术语"kinds"来引用这些服务类型。

表 1.1. 支持自动服务绑定的 Operator

服务绑定类型	Operator	API 版本	Kind
postgresql	CrunchyData Postgres	postgres-operator.crunchydata.com/v1beta1	PostgresCluster
mysql	Percona XtraDB 集群	pxc.percona.com/v1-9-0	PerconaXtraDBCluster
mongo	Percona MongoDB	psmdb.percona.com/v1-9-0	PerconaServerMongoDB



重要

- Red Hat build of Quarkus 3.8 支持 MongoDB Operator 作为技术预览提供，只适用于客户端。
- 如需红帽构建的 Quarkus 3.8 中支持的 Panache 扩展列表，请参阅 Quarkus 应用程序配置器 页面。

1.5.1. 自动数据源绑定

对于传统数据库，每当配置数据源时，都会启动自动绑定，如下所示：

```
quarkus.datasource.db-kind=postgresql
```

前面提到的配置以及存在 **quarkus-datasource**、**quarkus-jdbc-postgresql**、**quarkus-kubernetes** - kubernetes 和 **quarkus-kubernetes-service-binding** 等扩展，会导致为 **postgresql** 数据库类型创建 **ServiceBinding** 资源。

通过使用 Operator 资源的 **apiVersion** 和 **kind** 属性（与使用的 **postgresql** Operator 匹配），生成的 **ServiceBinding** 资源将服务或资源绑定到应用程序。

当您没有为数据库服务指定名称时，**db-kind** 属性的值将用作默认名称。

```
services:
- apiVersion: postgres-operator.crunchydata.com/v1beta1
  kind: PostgresCluster
  name: postgresql
```

指定数据源的名称，如下所示：

```
quarkus.datasource.fruits-db.db-kind=postgresql
```

生成的 **ServiceBinding** 中的服务如下所示：

```
services:
- apiVersion: postgres-operator.crunchydata.com/v1beta1
  kind: PostgresCluster
```

```
  name: fruits-db
```

同样，如果您使用 mysql，可以指定数据源的名称，如下所示：

```
quarkus.datasource.fruits-db.db-kind=mysql
```

生成的 服务 包含以下内容：

```
services:
- apiVersion: pxc.percona.com/v1-9-0
  kind: PerconaXtraDBCluster
  name: fruits-db
```

1.5.1.1. 自定义自动服务绑定

虽然开发了自动服务绑定功能来消除尽可能多的手动配置，但在某些情况下，您可能需要手动修改生成的 ServiceBinding 资源。

生成过程专门依赖于从应用程序中提取的信息以及支持的 Operator 知识，这些 Operator 可能无法反映集群中部署的内容。

生成的资源通常基于对常见服务类型支持的可绑定 Operator 的知识，以及为防止可能不匹配而开发的一系列约定，例如：

- 目标资源名称与数据源名称不匹配。
- 需要为该服务类型使用特定 Operator 而不是默认 Operator。
- 当用户需要使用非默认版本或最新的版本时，会发生版本冲突。

约定 ：

- 目标资源协调会根据 Operator 类型和服务类型建立。

- 默认情况下，目标资源名称与服务类型一致，如 postgresql、mysql 或 mongo。
- 如果是命名数据源，则使用数据源名称。
- 客户端的名称用于命名 mongo 客户端。

示例 1：名称不匹配

如果您需要修改生成的 ServiceBinding 来修复名称不匹配，请使用 quarkus.kubernetes-service-binding.services 属性，并将服务名称指定为服务键。

service 键 通常是服务的名称，例如，数据源的名称或 mongo 客户端的名称。当这个值不可用时，改为使用 postgresql、mysql 或 mongo 等数据源类型。

为了避免不同类型的服务之间命名冲突，请使用特定数据源类型（如 postgresql- *<person>* ） *作为服务密钥 添加前缀*。

以下示例演示了如何自定义 PostgresCluster 资源的 apiVersion 属性：

```
quarkus.datasource.db-kind=postgresql
quarkus.kubernetes-service-binding.services.postgresql.api-version=postgres-operator.crunchydata.com/v1beta2
```

示例 2：为数据源应用自定义名称

在示例 1 中，使用服务键 db-kind (postgresql)。在本实例中，使用规则后使用数据源名称(fruits-db)，因为数据源被命名。

以下示例显示，对于命名数据源，数据源名称用作目标资源的名称：

```
quarkus.datasource.fruits-db.db-kind=postgresql
```

这与以下配置相同：

```
quarkus.kubernetes-service-binding.services.fruits-db.api-version=postgres-operator.crunchydata.com/v1beta1
```

```
quarkus.kubernetes-service-binding.services.fruits-db.kind=PostgresCluster
quarkus.kubernetes-service-binding.services.fruits-db.name=fruits-db
```

其他资源

- 有关可用属性的更多信息，请参阅 Kubernetes 服务绑定规格 的工作负载投射 部分。

更新于 2024-05-10