



Red Hat Integration 2023.Q4

在 OpenShift 上安装和部署 Service Registry

安装、部署和配置 Service Registry 2.5

Red Hat Integration 2023.Q4 在 OpenShift 上安装和部署 Service Registry

安装、部署和配置 Service Registry 2.5

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

本指南介绍了如何在 AMQ Streams 或 PostgreSQL 数据库中数据存储选项在 OpenShift 上安装和部署 Service Registry。本指南还介绍了如何保护、配置和管理 Service Registry 部署，并提供 Service Registry 和 Service Registry Operator 的配置引用。

目录

前言	3
使开源包含更多	3
对红帽文档提供反馈	3
第 1 章 SERVICE REGISTRY OPERATOR 快速入门	4
1.1. QUICKSTART SERVICE REGISTRY OPERATOR 安装	4
1.2. QUICKSTART SERVICE REGISTRY 实例部署	4
第 2 章 在 OPENSIFT 上安装 SERVICE REGISTRY	6
2.1. 从 OPENSIFT OPERATORHUB 安装 SERVICE REGISTRY	6
第 3 章 在 AMQ STREAMS 中部署 SERVICE REGISTRY 存储	8
3.1. 从 OPENSIFT OPERATORHUB 安装 AMQ STREAMS	8
3.2. 在 OPENSIFT 中配置带有 KAFKA 存储的 SERVICE REGISTRY	9
3.3. 使用 TLS 安全性配置 KAFKA 存储	11
3.4. 使用 SCRAM 安全性配置 KAFKA 存储	14
3.5. 为 KAFKA 存储配置 OAUTH 身份验证	18
第 4 章 在 POSTGRESQL 数据库中部署 SERVICE REGISTRY 存储	20
4.1. 从 OPENSIFT OPERATORHUB 安装 POSTGRESQL 数据库	20
4.2. 在 OPENSIFT 中使用 POSTGRESQL 数据库存储配置 SERVICE REGISTRY	21
4.3. 备份 SERVICE REGISTRY POSTGRESQL 存储	22
4.4. 恢复 SERVICE REGISTRY POSTGRESQL 存储	23
第 5 章 保护 SERVICE REGISTRY 部署	24
5.1. 使用 RED HAT SINGLE SIGN-ON OPERATOR 保护 SERVICE REGISTRY	24
5.2. 使用 RED HAT SINGLE SIGN-ON 配置 SERVICE REGISTRY 身份验证和授权	28
5.3. SERVICE REGISTRY 身份验证和授权配置选项	31
5.4. 从 OPENSIFT 集群内部配置到 SERVICE REGISTRY 的 HTTPS 连接	36
5.5. 从 OPENSIFT 集群外部配置到 SERVICE REGISTRY 的 HTTPS 连接	37
第 6 章 配置和管理 SERVICE REGISTRY 部署	39
6.1. 在 OPENSIFT 中配置 SERVICE REGISTRY 健康检查	39
6.2. SERVICE REGISTRY 健康检查的环境变量	40
6.3. 管理 SERVICE REGISTRY 环境变量	41
6.4. 使用 PODTEMPLATE 配置 SERVICE REGISTRY 部署	42
6.5. 配置 SERVICE REGISTRY WEB 控制台	44
6.6. 配置 SERVICE REGISTRY 日志	45
6.7. 配置 SERVICE REGISTRY 事件源	45
第 7 章 SERVICE REGISTRY OPERATOR 配置参考	48
7.1. SERVICE REGISTRY 自定义资源	48
7.2. SERVICE REGISTRY CR SPEC	49
7.3. SERVICE REGISTRY CR 状态	54
7.4. SERVICE REGISTRY 受管资源	55
7.5. SERVICE REGISTRY OPERATOR 标签	55
第 8 章 SERVICE REGISTRY 配置参考	57
8.1. SERVICE REGISTRY 配置选项	57
附录 A. 使用您的订阅	67
访问您的帐户	67
激活订阅	67
下载 ZIP 和 TAR 文件	67

前言

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。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](#)。

对红帽文档提供反馈

我们感谢您对我们文档的反馈。

要改进，创建一个 JIRA 问题并描述您推荐的更改。提供尽可能多的详细信息，以便我们快速解决您的请求。

前提条件

- 您有一个红帽客户门户网站帐户。此帐户可让您登录到 Red Hat Jira Software 实例。如果您没有帐户，系统会提示您创建一个帐户。

流程

1. 单击以下链接：[创建问题](#)。
2. 在 **Summary** 文本框中输入问题的简短描述。
3. 在 **Description** 文本框中提供以下信息：
 - 找到此问题的页面的 URL。
 - 有关此问题的详细描述。
您可以将信息保留在任何其他字段中的默认值。
4. 点 **Create** 将 JIRA 问题提交到文档团队。

感谢您花时间来提供反馈。

第 1 章 SERVICE REGISTRY OPERATOR 快速入门

您可以使用自定义资源定义(CRD)在命令行中快速安装 Service Registry Operator。

Quickstart 示例使用 SQL 数据库中的存储部署 Service Registry 实例：

- [第 1.1 节 “Quickstart Service Registry Operator 安装”](#)
- [第 1.2 节 “Quickstart Service Registry 实例部署”](#)



注意

生产环境的建议安装选项是 OpenShift OperatorHub。推荐的存储选项是 SQL 数据库，用于性能、稳定性和数据管理。

1.1. QUICKSTART SERVICE REGISTRY OPERATOR 安装

您可以使用下载的安装文件和示例 CRD，在命令行中快速安装和部署 Service Registry Operator，而无需 Operator Lifecycle Manager。

先决条件

- 您使用管理员访问权限登录到 OpenShift 集群。
- 已安装 OpenShift **oc** 命令行客户端。如需了解更多详细信息，请参阅 [OpenShift CLI 文档](#)。

流程

1. 浏览 [Red Hat Software Downloads](#)，选择产品版本，并下载 Service Registry CRD **.zip** 文件中的示例。
2. 提取下载的 CRD **.zip** 文件并改为 **apicurio-registry-install-examples** 目录。
3. 为 Service Registry Operator 安装创建一个 OpenShift 项目，例如：

```
export NAMESPACE="apicurio-registry"
oc new-project "$NAMESPACE"
```

4. 输入以下命令在 **install/install.yaml** 文件中应用示例 CRD：

```
cat install/install.yaml | sed "s/apicurio-registry-operator-namespace/$NAMESPACE/g" | oc
apply -f -
```

5. 输入 **oc get deployment** 以检查 Service Registry Operator 的就绪情况。例如，输出应如下：

```
NAME                READY  UP-TO-DATE  AVAILABLE  AGE
apicurio-registry-operator  1/1    1           1          XmYs
```

1.2. QUICKSTART SERVICE REGISTRY 实例部署

要创建 Service Registry 实例部署，请使用 SQL 数据库存储选项连接到现有的 PostgreSQL 数据库。

先决条件

- 确保已安装 Service Registry Operator。
- 您有一个可从 OpenShift 集群访问的 PostgreSQL 数据库。

流程

1. 在编辑器中打开 `examples/apicurioregistry_sql_cr.yaml` 文件，并查看 **ApicurioRegistry** 自定义资源(CR)：

SQL 存储的 CR 示例

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-sql
spec:
  configuration:
    persistence: "sql"
    sql:
      dataSource:
        url: "jdbc:postgresql://<service name>.<namespace>.svc:5432/<database name>"
        userName: "postgres"
        password: "<password>" # Optional
```

2. 在 **dataSource** 部分中，将示例设置替换为您的数据库连接详情。例如：

```
dataSource:
  url: "jdbc:postgresql://postgresql.apicurio-registry.svc:5432/registry"
  userName: "pgadmin"
  password: "pgpass"
```

3. 输入以下命令在带有 Service Registry Operator 的命名空间中应用更新的 **ApicurioRegistry** CR，并等待 Service Registry 实例部署：

```
oc project "$NAMESPACE"
oc apply -f ./examples/apicurioregistry_sql_cr.yaml
```

4. 输入 **oc get deployment** 来检查 Service Registry 实例的就绪情况。例如，输出应如下：

```
NAME                                READY UP-TO-DATE AVAILABLE AGE
example-apicurioregistry-sql-deployment 1/1 1 1 XmYs
```

5. 输入 **oc get routes** 来获取 **HOST/PORT** URL，以在浏览器中启动 Service Registry web 控制台。例如：

```
example-apicurioregistry-sql.apicurio-registry.router-
default.apps.mycluster.myorg.mycompany.com
```

第 2 章 在 OPENSIFT 上安装 SERVICE REGISTRY

本章论述了如何在 OpenShift Container Platform 上安装 Service Registry :

- [第 2.1 节 “从 OpenShift OperatorHub 安装 Service Registry”](#)

先决条件

- 阅读 [Service Registry 用户指南](#)中的 简介。

2.1. 从 OPENSIFT OPERATORHUB 安装 SERVICE REGISTRY

您可以从 OperatorHub 在 OpenShift 集群上安装 Service Registry Operator。OperatorHub 由 OpenShift Container Platform Web 控制台获得，为集群管理员提供了一个界面，以发现和安装 Operator。如需了解更多详细信息，[请参阅了解 OperatorHub](#)。



注意

您可以根据您的环境安装多个 Service Registry 实例。实例数量取决于存储在 Service Registry 中的工件的数量和类型，以及您选择的存储选项。

先决条件

- 您必须具有集群管理员对 OpenShift 集群的访问权限。

流程

1. 在 OpenShift Container Platform Web 控制台中，使用具有集群管理员特权的帐户登录。
2. 创建新的 OpenShift 项目：
 - a. 在左侧导航菜单中，单击 **Home**、**Project**，然后单击 **Create Project**。
 - b. 输入项目名称，如 **my-project**，再单击 **Create**。
3. 在左侧导航菜单中，点 **Operators**，然后点 **OperatorHub**。
4. 在 **Filter by keyword** 文本框中，输入 **registry** 来查找 **Red Hat Integration - Service Registry Operator**。
5. 阅读有关 Operator 的信息，点 **Install** 显示 Operator 订阅页面。
6. 选择您的订阅设置，例如：
 - **更新频道**：选择以下之一：
 - **2.x**：包括所有次要和补丁更新，如 2.3.0 和 2.0.3。例如，2.0.x 上的安装将升级到 2.3.x。
 - **2.0.x**：仅包含补丁更新，如 2.0.1 和 2.0.2。例如，2.0.x 上的安装将忽略 2.3.x。
 - **安装模式**：选择以下之一：
 - **集群上的所有命名空间（默认）**
 - **一个特定命名空间，然后是 my-project**

- **批准策略**：选择 **Automatic** 或 **Manual**

7. 点 **Install**，等待片刻，直到 Operator 准备就绪为止。

其他资源

- [将 Operator 添加到 OpenShift 集群](#)
- [GitHub 中的 Apicurio Registry Operator 社区](#)

第 3 章 在 AMQ STREAMS 中部署 SERVICE REGISTRY 存储

本章论述了如何在 AMQ Streams 中安装和配置 Service Registry 数据存储。

- [第 3.1 节 “从 OpenShift OperatorHub 安装 AMQ Streams”](#)
- [第 3.2 节 “在 OpenShift 中配置带有 Kafka 存储的 Service Registry”](#)
- [第 3.3 节 “使用 TLS 安全性配置 Kafka 存储”](#)
- [第 3.4 节 “使用 SCRAM 安全性配置 Kafka 存储”](#)
- [第 3.5 节 “为 Kafka 存储配置 OAuth 身份验证”](#)

先决条件

- [第 2 章 在 OpenShift 上安装 Service Registry](#)

3.1. 从 OPENSIFT OPERATORHUB 安装 AMQ STREAMS

如果您还没有安装 AMQ Streams，您可以从 OperatorHub 在 OpenShift 集群上安装 AMQ Streams Operator。OperatorHub 由 OpenShift Container Platform Web 控制台获得，为集群管理员提供了一个界面，以发现和安装 Operator。如需了解更多详细信息，[请参阅了解 OperatorHub](#)。

先决条件

- 您必须具有集群管理员对 OpenShift 集群的访问权限
- 有关安装 [AMQ Streams](#) 的详细信息，[请参阅在 OpenShift 中部署和管理 AMQ Streams](#)。本节展示了使用 OpenShift OperatorHub 安装的简单示例。

流程

1. 在 OpenShift Container Platform Web 控制台中，使用具有集群管理员特权的帐户登录。
2. 切换到您要在其中安装 AMQ Streams 的 OpenShift 项目。例如，从 **Project** 下拉菜单中选择 **my-project**。
3. 在左侧导航菜单中，点 **Operators**，然后点 **OperatorHub**。
4. 在 **Filter by keyword** 文本框中，输入 **AMQ Streams** 来查找 **Red Hat Integration - AMQ Streams Operator**。
5. 阅读有关 Operator 的信息，点 **Install** 显示 Operator 订阅页面。
6. 选择您的订阅设置，例如：
 - **更新频道**，然后 **amq-streams-2.5.x**
 - **安装模式**：选择以下之一：
 - **集群上的所有命名空间（默认）**
 - **A specific namespace on the cluster > my-project**
 - **批准策略**：选择 **Automatic** 或 **Manual**

- 点 **Install**，等待片刻，直到 Operator 准备就绪为止。

其他资源

- [将 Operator 添加到 OpenShift 集群](#)
- [在 OpenShift 中部署和管理 AMQ Streams](#)

3.2. 在 OPENSIFT 中配置带有 KAFKA 存储的 SERVICE REGISTRY

本节介绍如何在 OpenShift 上使用 AMQ Streams 为 Service Registry 配置基于 Kafka 的存储。**kafkasql** 存储选项使用带有内存 H2 数据库的 Kafka 存储进行缓存。当为 OpenShift 上的 Kafka 集群配置了 **持久性存储**时，此存储选项适用于生产环境。

您可以在现有 Kafka 集群中安装 Service Registry，或根据您的环境创建新 Kafka 集群。

先决条件

- 您必须具有具有集群管理员访问权限的 OpenShift 集群。
- 您必须已安装了 Service Registry。请参阅 [第 2 章 在 OpenShift 上安装 Service Registry](#)。
- 您必须已安装了 AMQ Streams。请参阅 [第 3.1 节 “从 OpenShift OperatorHub 安装 AMQ Streams”](#)。

流程

1. 在 OpenShift Container Platform Web 控制台中，使用具有集群管理员特权的帐户登录。
2. 如果您还没有配置 Kafka 集群，请使用 AMQ Streams 创建新的 Kafka 集群。例如，在 OpenShift OperatorHub 中：
 - a. 点 **Installed Operators**，然后点 **Red Hat Integration - AMQ Streams**
 - b. 在 **Provided APIs** 下，然后点 **Kafka**，点 **Create Instance** 创建新的 Kafka 集群。
 - c. 根据需要编辑自定义资源定义，然后点 **Create**。



警告

默认示例创建了一个集群，这个集群带有 3 个 Zookeeper 节点，以及 3 个带有 **临时存储**的 Kafka 节点。此临时存储仅适用于开发和测试，不适用于生产环境。如需了解更多详细信息，[请参阅在 OpenShift 中部署和管理 AMQ Streams](#)。

3. 集群就绪后，点 **Provided APIs > Kafka > my-cluster > YAML**。
4. 在 **status** 块中，复制 **bootstrapServers** 值，稍后您将使用它来部署 Service Registry。例如：

```
status:
```

```

...
conditions:
...
listeners:
  - addresses:
    - host: my-cluster-kafka-bootstrap.my-project.svc
      port: 9092
    bootstrapServers: 'my-cluster-kafka-bootstrap.my-project.svc:9092'
    type: plain
...

```

5. 点 **Installed Operators > Red Hat Integration - Service Registry > ApicurioRegistry > Create ApicurioRegistry**。

6. 粘贴以下自定义资源定义，但使用您之前复制的 **bootstrapServers** 值：

```

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql
spec:
  configuration:
    persistence: 'kafkasql'
    kafkasql:
      bootstrapServers: 'my-cluster-kafka-bootstrap.my-project.svc:9092'

```

7. 点 **Create** 并等待 OpenShift 上创建 Service Registry 路由。

8. 点 **Networking > Route** 访问 Service Registry web 控制台的新路由。例如：

```
http://example-apicurioregistry-kafkasql.my-project.my-domain-name.com/
```

9. 要配置 Service Registry 用来存储数据的 Kafka 主题，请点 **Installed Operators > Red Hat Integration - AMQ Streams > Provided APIs > Kafka Topic > kafkasql-journal > YAML**。例如：

```

apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: kafkasql-journal
  labels:
    strimzi.io/cluster: my-cluster
  namespace: ...
spec:
  partitions: 3
  replicas: 3
  config:
    cleanup.policy: compact

```



警告

您必须使用压缩清理策略配置 Service Registry 使用的 Kafka 主题（默认为 `kafkasql-journal`），否则可能会出现数据丢失。

其他资源

- 有关使用 AMQ Streams 创建 Kafka 集群和主题的详情，请参阅在 [OpenShift 中部署和管理 AMQ Streams](#)。

3.3. 使用 TLS 安全性配置 KAFKA 存储

您可以将 AMQ Streams Operator 和 Service Registry Operator 配置为使用加密的传输层安全(TLS)连接。

先决条件

- 已使用 OperatorHub 或命令行安装 Service Registry Operator。
- 已安装 AMQ Streams Operator 或可以从 OpenShift 集群访问 Kafka。



注意

本节假设 AMQ Streams Operator 可用，但您可以使用任何 Kafka 部署。在这种情况下，您必须手动创建 Service Registry Operator 期望的 Openshift secret。

流程

1. 在 OpenShift Web 控制台中，点 **Installed Operators**，选择 **AMQ Streams Operator** 详情，然后选择 **Kafka** 选项卡。
2. 点 **Create Kafka** 为 Service Registry 存储置备新的 Kafka 集群。
3. 配置 **authorization** 和 **tls** 字段，以将 TLS 身份验证用于 Kafka 集群，例如：

```

apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: registry-example-kafkasql-tls
  # Change or remove the explicit namespace
spec:
  kafka:
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      log.message.format.version: '2.7'
      inter.broker.protocol.version: '2.7'
    version: 2.7.0

```

```

storage:
  type: ephemeral
replicas: 3
listeners:
  - name: tls
    port: 9093
    type: internal
    tls: true
    authentication:
      type: tls
  authorization:
    type: simple
entityOperator:
  topicOperator: {}
  userOperator: {}
zookeeper:
  storage:
    type: ephemeral
  replicas: 3

```

Service Registry 自动创建的默认 Kafka 主题名称为 **kafkasql-journal**。您可以通过设置环境变量来覆盖此行为或默认主题名称。默认值如下：

- **REGISTRY_KAFKASQL_TOPIC_AUTO_CREATE=true**
- **REGISTRY_KAFKASQL_TOPIC=kafkasql-journal**

如果您决定不手动创建 Kafka 主题，请跳过下一步。

4. 点 **Kafka Topic** 选项卡，然后点 **Create Kafka Topic** 创建 **kafkasql-journal** 主题：

```

apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaTopic
metadata:
  name: kafkasql-journal
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-tls
spec:
  partitions: 2
  replicas: 1
  config:
    cleanup.policy: compact

```

5. 创建 **Kafka User** 资源，以配置 Service Registry 用户的身份验证和授权。您可以在 **metadata** 部分中指定用户名，或者使用默认的 **my-user**。

```

apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-tls
spec:
  authentication:

```

```

type: tls
authorization:
  acls:
    - operation: All
      resource:
        name: '*'
        patternType: literal
        type: topic
    - operation: All
      resource:
        name: '*'
        patternType: literal
        type: cluster
    - operation: All
      resource:
        name: '*'
        patternType: literal
        type: transactionalId
    - operation: All
      resource:
        name: '*'
        patternType: literal
        type: group
type: simple

```



注意

这个简单示例假设 admin 权限并自动创建 Kafka 主题。您必须为 Service Registry 所需的主题和资源配置 **授权** 部分。

以下示例显示了手动创建 Kafka 主题时所需的最低配置：

```

...
authorization:
  acls:
    - operations:
      - Read
      - Write
    resource:
      name: kafkasql-journal
      patternType: literal
      type: topic
    - operations:
      - Read
      - Write
    resource:
      name: apicurio-registry-
      patternType: prefix
      type: group
type: simple

```

6. 点 **Workloads**，然后 **Secrets** 来查找 AMQ Streams 为 Service Registry 创建的两个 secret 来连接到 Kafka 集群：

- **my-cluster-cluster-ca-cert** - 包含 Kafka 集群的 PKCS12 信任存储

- **my-user** - 包含用户的密钥存储



注意

secret 的名称可能会因集群或用户名而异。

7. 如果手动创建 secret，它们必须包含以下键值对：

- **my-cluster-ca-cert**
 - **ca.p12** - 信任存储，格式为 PKCS12
 - **ca.password** - truststore password
- **my-user**
 - **user.p12** - 密钥存储（以 PKCS12 格式）
 - **user.password** - keystore password

8. 配置以下示例配置，以部署 Service Registry。

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql-tls
spec:
  configuration:
    persistence: "kafkasql"
    kafkasql:
      bootstrapServers: "my-cluster-kafka-bootstrap.registry-example-kafkasql-tls.svc:9093"
      security:
        tls:
          keystoreSecretName: my-user
          truststoreSecretName: my-cluster-cluster-ca-cert
```



重要

您必须使用不同的 **bootstrapServers** 地址，而不是在不安全的用例中。该地址必须支持 TLS 连接，并在 **type: tls** 字段下的指定 **Kafka** 资源中找到。

3.4. 使用 SCRAM 安全性配置 KAFKA 存储

您可以将 AMQ Streams Operator 和 Service Registry Operator 配置为使用 Kafka 集群的 Salted Challenge Response Authentication Mechanism (SCRAM-SHA-512)。

先决条件

- 已使用 OperatorHub 或命令行安装 Service Registry Operator。
- 已安装 AMQ Streams Operator 或可以从 OpenShift 集群访问 Kafka。



注意

本节假设 AMQ Streams Operator 可用，但您可以使用任何 Kafka 部署。在这种情况下，您必须手动创建 Service Registry Operator 期望的 Openshift secret。

流程

1. 在 OpenShift Web 控制台中，点 **Installed Operators**，选择 **AMQ Streams Operator** 详情，然后选择 **Kafka** 选项卡。
2. 点 **Create Kafka** 为 Service Registry 存储置备新的 Kafka 集群。
3. 将 **authorization** 和 **tls** 字段配置为对 Kafka 集群使用 SCRAM-SHA-512 身份验证，例如：

```

apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: registry-example-kafkasql-scram
  # Change or remove the explicit namespace
spec:
  kafka:
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      log.message.format.version: '2.7'
      inter.broker.protocol.version: '2.7'
    version: 2.7.0
    storage:
      type: ephemeral
    replicas: 3
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: scram-sha-512
    authorization:
      type: simple
    entityOperator:
      topicOperator: {}
      userOperator: {}
    zookeeper:
      storage:
        type: ephemeral
      replicas: 3
  
```

Service Registry 自动创建的默认 Kafka 主题名称为 **kafkasql-journal**。您可以通过设置环境变量来覆盖此行为或默认主题名称。默认值如下：

- **REGISTRY_KAFKASQL_TOPIC_AUTO_CREATE=true**
- **REGISTRY_KAFKASQL_TOPIC=kafkasql-journal**

如果您决定不手动创建 Kafka 主题，请跳过下一步。

4. 点 **Kafka Topic** 选项卡，然后点 **Create Kafka Topic** 创建 **kafkasql-journal** 主题：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaTopic
metadata:
  name: kafkasql-journal
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-scrum
spec:
  partitions: 2
  replicas: 1
  config:
    cleanup.policy: compact
```

5. 创建 **Kafka User** 资源，为 Service Registry 用户配置 SCRAM 身份验证和授权。您可以在 **metadata** 部分中指定用户名，或者使用默认的 **my-user**。

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-scrum
spec:
  authentication:
    type: scram-sha-512
  authorization:
    acls:
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: topic
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: cluster
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: transactionalId
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: group
    type: simple
```

**注意**

这个简单示例假设 admin 权限并自动创建 Kafka 主题。您必须为 Service Registry 所需的主题和资源配置 **授权** 部分。

以下示例显示了手动创建 Kafka 主题时所需的最低配置：

```
...
authorization:
  acls:
    - operations:
      - Read
      - Write
    resource:
      name: kafkasql-journal
      patternType: literal
      type: topic
    - operations:
      - Read
      - Write
    resource:
      name: apicurio-registry-
      patternType: prefix
      type: group
  type: simple
```

6. 点 **Workloads**，然后 **Secrets** 来查找 AMQ Streams 为 Service Registry 创建的两个 secret 来连接到 Kafka 集群：

- **my-cluster-cluster-ca-cert** - 包含 Kafka 集群的 PKCS12 信任存储
- **my-user** - 包含用户的密钥存储

**注意**

secret 的名称可能会因集群或用户名而异。

7. 如果手动创建 secret，它们必须包含以下键值对：

- **my-cluster-ca-cert**
 - **ca.p12** - PKCS12 格式的信任存储
 - **ca.password** - truststore password
- **my-user**
 - **password** - 用户密码

8. 配置以下示例设置来部署 Service Registry：

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql-scrum
```

```
spec:
  configuration:
    persistence: "kafkasql"
    kafkasql:
      bootstrapServers: "my-cluster-kafka-bootstrap.registry-example-kafkasql-
scram.svc:9093"
    security:
      scram:
        truststoreSecretName: my-cluster-cluster-ca-cert
        user: my-user
        passwordSecretName: my-user
```



重要

您必须使用不同的 **bootstrapServers** 地址，而不是在不安全的用例中。该地址必须支持 TLS 连接，并在 **type: tls** 字段下的指定 **Kafka** 资源中找到。

3.5. 为 KAFKA 存储配置 OAUTH 身份验证

在 AMQ Streams 中使用基于 Kafka 的存储时，Service Registry 支持访问需要 OAuth 身份验证的 Kafka 集群。要启用此支持，您必须在 Service Registry 部署中设置一些环境变量。

当您设置这些环境变量时，Service Registry 中的 Kafka producer 和消费者应用程序将使用此配置通过 OAuth 向 Kafka 集群进行身份验证。

先决条件

- 您必须在 AMQ Streams 中配置了基于 Kafka 的 Service Registry 数据存储。请参阅 [第 3.2 节“在 OpenShift 中配置带有 Kafka 存储的 Service Registry”](#)。

流程

- 在 Service Registry 部署中设置以下环境变量：

环境变量	描述	默认值
ENABLE_KAFKA_SASL	为 Kafka 中的 Service Registry 存储启用 SASL OAuth 身份验证。您必须将此变量设置为 true ，以便其他变量生效。	false
CLIENT_ID	用于向 Kafka 进行身份验证的客户端 ID。	-
CLIENT_SECRET	用于向 Kafka 进行身份验证的客户端 secret。	-
OAuth_TOKEN_ENDPOINT_URI	OAuth 身份服务器的 URL。	http://localhost:8090

其他资源

- 有关如何在 OpenShift 上设置 Service Registry 环境变量的示例，请参阅 [第 6.1 节 “在 OpenShift 中配置 Service Registry 健康检查”](#)

第 4 章 在 POSTGRESQL 数据库中部署 SERVICE REGISTRY 存储

本章介绍了如何在 PostgreSQL 数据库中安装、配置和管理 Service Registry 数据存储。

- [第 4.1 节 “从 OpenShift OperatorHub 安装 PostgreSQL 数据库”](#)
- [第 4.2 节 “在 OpenShift 中使用 PostgreSQL 数据库存储配置 Service Registry”](#)
- [第 4.3 节 “备份 Service Registry PostgreSQL 存储”](#)
- [第 4.4 节 “恢复 Service Registry PostgreSQL 存储”](#)

先决条件

- [第 2 章 在 OpenShift 上安装 Service Registry](#)

4.1. 从 OPENSIFT OPERATORHUB 安装 POSTGRESQL 数据库

如果您还没有安装 PostgreSQL 数据库 Operator，您可以从 OperatorHub 在 OpenShift 集群上安装 PostgreSQL Operator。OperatorHub 由 OpenShift Container Platform Web 控制台获得，为集群管理员提供了一个界面，以发现和安装 Operator。如需了解更多详细信息，[请参阅了解 OperatorHub](#)。

先决条件

- 您必须具有集群管理员对 OpenShift 集群的访问权限。

流程

1. 在 OpenShift Container Platform Web 控制台中，使用具有集群管理员特权的帐户登录。
2. 切换到要在其中安装 PostgreSQL Operator 的 OpenShift 项目。例如，从 **Project** 下拉菜单中选择 **my-project**。
3. 在左侧导航菜单中，点 **Operators**，然后点 **OperatorHub**。
4. 在 **Filter by keyword** 文本框中，输入 **PostgreSQL** 以查找适合您的环境的 Operator，例如，**Crunchy PostgreSQL for OpenShift**。
5. 阅读有关 Operator 的信息，点 **Install** 显示 Operator 订阅页面。
6. 选择您的订阅设置，例如：
 - **更新频道**：stable
 - **Installation Mode**：集群中的特定命名空间，然后是 my-project
 - **批准策略**：选择 **Automatic** 或 **Manual**
7. 点 **Install**，等待片刻，直到 Operator 准备就绪为止。



重要

您必须从所选 PostgreSQL Operator 读取文档，以了解有关如何创建和管理数据库的详细信息。

其他资源

- [将 Operator 添加到 OpenShift 集群](#)
- [Crunchy PostgreSQL Operator QuickStart](#)

4.2. 在 OPENSIFT 中使用 POSTGRESQL 数据库存储配置 SERVICE REGISTRY

本节介绍如何使用 PostgreSQL 数据库 Operator 为 OpenShift 上的 Service Registry 配置存储。您可以在现有数据库中安装 Service Registry，或根据您的环境创建新数据库。本节演示了一个使用 PostgreSQL Operator by Dev4Ddevs.com 的简单示例。

先决条件

- 您必须具有具有集群管理员访问权限的 OpenShift 集群。
- 您必须已安装了 Service Registry。请参阅 [第 2 章 在 OpenShift 上安装 Service Registry](#)。
- 您必须已在 OpenShift 上安装 PostgreSQL Operator。例如，请参阅 [第 4.1 节 “从 OpenShift OperatorHub 安装 PostgreSQL 数据库”](#)。

流程

1. 在 OpenShift Container Platform Web 控制台中，使用具有集群管理员特权的帐户登录。
2. 切换到安装 Service Registry 和 PostgreSQL Operator 的 OpenShift 项目。例如，从 **Project** 下拉菜单中选择 **my-project**。
3. 为您的 Service Registry 存储创建一个 PostgreSQL 数据库。例如，单击 **Installed Operators**，**PostgreSQL Operator by Dev4Ddevs.com**，然后单击 **Create database**。
4. 点 **YAML** 并编辑数据库设置，如下所示：
 - **名称**：将值改为 **registry**
 - **Image**：将值改为 **centos/postgresql-12-centos7**
5. 根据您的环境，根据需要编辑任何其他数据库设置，例如：

```
apiVersion: postgresql.dev4devs.com/v1alpha1
kind: Database
metadata:
  name: registry
  namespace: my-project
spec:
  databaseCpu: 30m
  databaseCpuLimit: 60m
  databaseMemoryLimit: 512Mi
  databaseMemoryRequest: 128Mi
  databaseName: example
  databaseNameKeyEnvVar: POSTGRESQL_DATABASE
  databasePassword: postgres
  databasePasswordKeyEnvVar: POSTGRESQL_PASSWORD
  databaseStorageRequest: 1Gi
  databaseUser: postgres
```

```
databaseUserKeyEnvVar: POSTGRESQL_USER
image: centos/postgresql-12-centos7
size: 1
```

- 点 **Create**，然后等待数据库创建完毕。
- 点 **Installed Operators > Red Hat Integration - Service Registry > ApicurioRegistry > Create ApicurioRegistry**。
- 粘贴以下自定义资源定义，并编辑数据库 **url** 和凭证的值以匹配您的环境：

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-sql
spec:
  configuration:
    persistence: 'sql'
    sql:
      dataSource:
        url: 'jdbc:postgresql://<service name>.<namespace>.svc:5432/<database name>'
        # e.g. url: 'jdbc:postgresql://acid-minimal-cluster.my-project.svc:5432/registry'
        userName: 'postgres'
        password: '<password>' # Optional
```

- 点 **Create** 并等待 OpenShift 上创建 Service Registry 路由。
- 点 **Networking > Route** 访问 Service Registry web 控制台的新路由。例如：

```
http://example-apicurioregistry-sql.my-project.my-domain-name.com/
```

其他资源

- [Crunchy PostgreSQL Operator QuickStart](#)
- [Apicurio Registry Operator QuickStart](#)

4.3. 备份 SERVICE REGISTRY POSTGRESQL 存储

在 PostgreSQL 数据库中使用存储时，您必须确保 Service Registry 存储的数据会定期备份。

SQL Dump 是可用于任何 PostgreSQL 安装的简单流程。这使用 **pg_dump** 程序生成带有 SQL 命令的文件，您可以使用它们以与转储时相同状态重新创建数据库。

pg_dump 是一个常规 PostgreSQL 客户端应用程序，您可以从有权访问数据库的任何远程主机执行。与其他客户端一样，可以执行的操作仅限于用户权限。

流程

- 使用 **pg_dump** 命令将输出重定向到文件中：

```
$ pg_dump dbname > dumpfile
```

您可以使用 **-h host** 和 **-p port** 选项指定 **pg_dump** 连接到的数据库。

- 您可以使用压缩工具（如 gzip）减少大型转储文件，例如：

```
$ pg_dump dbname | gzip > filename.gz
```

其他资源

- 有关客户端身份验证的详情，请查看 [PostgreSQL 文档](#)。
- 有关导入和导出 registry 内容的详情，请参阅使用 [REST API 管理 Service Registry 内容](#)。

4.4. 恢复 SERVICE REGISTRY POSTGRESQL 存储

您可以使用 `psql` 程序恢复 `pg_dump` 创建的 SQL 转储文件。

先决条件

- 您必须已使用 `pg_dump` 备份 PostgreSQL database。请参阅 [第 4.3 节“备份 Service Registry PostgreSQL 存储”](#)。
- 所有拥有对象或对转储数据库中对象具有权限的用户都必须已存在。

流程

1. 运行以下命令来创建数据库：

```
$ createdb -T template0 dbname
```

2. 输入以下命令恢复 SQL 转储

```
$ psql dbname < dumpfile
```

3. 在每个数据库上运行 `ANALYZE`，以便查询优化器具有有用的统计信息。

第 5 章 保护 SERVICE REGISTRY 部署

本章介绍了如何在 OpenShift 上为 Service Registry 部署配置安全设置：

- [第 5.1 节 “使用 Red Hat Single Sign-On Operator 保护 Service Registry”](#)
- [第 5.2 节 “使用 Red Hat Single Sign-On 配置 Service Registry 身份验证和授权”](#)
- [第 5.3 节 “Service Registry 身份验证和授权配置选项”](#)
- [第 5.4 节 “从 OpenShift 集群内部配置到 Service Registry 的 HTTPS 连接”](#)
- [第 5.5 节 “从 OpenShift 集群外部配置到 Service Registry 的 HTTPS 连接”](#)

Service Registry 使用基于 OpenID Connect (OIDC) 或 HTTP 基本的 Red Hat Single Sign-On 提供身份验证和授权。您可以使用 Red Hat Single Sign-On Operator 自动配置所需的设置，或者在 Red Hat Single Sign-On 和 Service Registry 中手动配置它们。

Service Registry 使用 Red Hat Single Sign-On 为 Service Registry Web 控制台和核心 REST API 提供基于角色的身份验证和授权。Service Registry 在 schema 或 API 级别上提供基于内容的授权，其中只有工件创建者具有写入访问权限。您还可以从 OpenShift 集群内部或外部配置到 Service Registry 的 HTTPS 连接。

其他资源

- 有关 Java 客户端应用程序安全配置的详情，请查看以下内容：
 - [Service Registry Java 客户端配置](#)
 - [Service Registry serializer/deserializer 配置](#)

5.1. 使用 RED HAT SINGLE SIGN-ON OPERATOR 保护 SERVICE REGISTRY

以下流程演示了如何配置由 Red Hat Single Sign-On 保护的 Service Registry REST API 和 Web 控制台。

Service Registry 支持以下用户角色：

表 5.1. Service Registry 用户角色

名称	功能
sr-admin	完全访问权限，没有限制。
sr-developer	创建工件并配置工件规则。无法修改全局规则、执行导入/导出或使用 /admin REST API 端点。
sr-readonly	仅查看和搜索。无法修改工件或规则，执行导入/导出或使用 /admin REST API 端点。



注意

ApicurioRegistry CRD 中有一个相关的配置选项，可用于将 Web 控制台设置为只读模式。但是，此配置不会影响 REST API。

先决条件

- 您必须已安装了 Service Registry Operator。
- 您必须安装 Red Hat Single Sign-On Operator，或者可从 OpenShift 集群访问 Red Hat Single Sign-On。



重要

此流程中的示例配置仅用于开发和测试。为了保持流程简单，它不会在生产环境中使用 HTTPS 和其他推荐的防御。如需了解更多详细信息，请参阅 Red Hat Single Sign-On 文档。

流程

1. 在 OpenShift Web 控制台中，点 **Installed Operators** 和 **Red Hat Single Sign-On Operator**，然后点 **Keycloak** 选项卡。
2. 点 **Create Keycloak** 置备一个新的 Red Hat Single Sign-On 实例来保护 Service Registry 部署。您可以使用默认值，例如：

```

apiVersion: keycloak.org/v1alpha1
kind: Keycloak
metadata:
  name: example-keycloak
  labels:
    app: sso
spec:
  instances: 1
  externalAccess:
    enabled: True
  podDisruptionBudget:
    enabled: True

```

3. 等待实例创建好，然后单击 **Networking**，然后单击 **Routes** 以访问 **keycloak** 实例的新路由。
4. 点 **Location URL**，并复制显示的 URL 值，以便在部署 Service Registry 时使用。
5. 点 **Installed Operators** 和 **Red Hat Single Sign-On Operator**，然后点 **Keycloak Realm** 选项卡，然后点 **Create Keycloak Realm** 来创建 **registry** 示例域：

```

apiVersion: keycloak.org/v1alpha1
kind: KeycloakRealm
metadata:
  name: registry-keycloakrealm
  labels:
    app: registry
spec:
  instanceSelector:
    matchLabels:

```

```
app: sso
realm:
  displayName: Registry
  enabled: true
  id: registry
  realm: registry
  sslRequired: none
  roles:
    realm:
      - name: sr-admin
      - name: sr-developer
      - name: sr-readonly
  clients:
    - clientId: registry-client-ui
      implicitFlowEnabled: true
      redirectUris:
        - '*'
      standardFlowEnabled: true
      webOrigins:
        - '*'
      publicClient: true
    - clientId: registry-client-api
      implicitFlowEnabled: true
      redirectUris:
        - '*'
      standardFlowEnabled: true
      webOrigins:
        - '*'
      publicClient: true
  users:
    - credentials:
        - temporary: false
          type: password
          value: changeme
      enabled: true
      realmRoles:
        - sr-admin
      username: registry-admin
    - credentials:
        - temporary: false
          type: password
          value: changeme
      enabled: true
      realmRoles:
        - sr-developer
      username: registry-developer
    - credentials:
        - temporary: false
          type: password
          value: changeme
      enabled: true
      realmRoles:
        - sr-readonly
      username: registry-user
```



重要

如果要部署到生产环境，则必须使用适合您的环境的值自定义此 **KeycloakRealm** 资源。您还可以使用 Red Hat Single Sign-On Web 控制台创建和管理域。

6. 如果您的集群没有配置有效的 HTTPS 证书，您可以创建以下 HTTP **Service** 和 **Ingress** 资源作为临时临时解决方案：

- a. 点 **Networking**，然后点 **Create Service**，使用以下示例点 **Create Service**：

```

apiVersion: v1
kind: Service
metadata:
  name: keycloak-http
  labels:
    app: keycloak
spec:
  ports:
    - name: keycloak-http
      protocol: TCP
      port: 8080
      targetPort: 8080
  selector:
    app: keycloak
    component: keycloak
  type: ClusterIP
  sessionAffinity: None
status:
  loadBalancer: {}

```

- b. 点 **Networking**，然后点 **Ingresses**，使用以下示例点 **Create Ingress**：

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: keycloak-http
  labels:
    app: keycloak
spec:
  rules:
    - host: KEYCLOAK_HTTP_HOST
      http:
        paths:
          - path: /
            pathType: ImplementationSpecific
            backend:
              service:
                name: keycloak-http
                port:
                  number: 8080

```

修改 **主机** 值，以创建可由 Service Registry 用户访问的路由，并使用它而不是 Red Hat Single Sign-On Operator 创建的 HTTPS 路由。

- 点 Service Registry Operator, 在 ApicurioRegistry 选项卡中, 点 Create ApicurioRegistry, 使用以下示例, 使用您的实际值替换 keycloak 部分。

```

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql-keycloak
spec:
  configuration:
    security:
      keycloak:
        url: "http://keycloak-http-<namespace>.apps.<cluster host>"
        # ^ Required
        # Use an HTTP URL in development.
        realm: "registry"
        # apiClientId: "registry-client-api"
        # ^ Optional (default value)
        # uiClientId: "registry-client-ui"
        # ^ Optional (default value)
      persistence: 'kafkasql'
      kafkasql:
        bootstrapServers: '<my-cluster>-kafka-bootstrap.<my-namespace>.svc:9092'

```

5.2. 使用 RED HAT SINGLE SIGN-ON 配置 SERVICE REGISTRY 身份验证和授权

本节介绍如何使用 Red Hat Single Sign-On 为 Service Registry 手动配置身份验证和授权选项。



注意

或者, 有关如何自动配置这些设置的详情, 请参考 [第 5.1 节 “使用 Red Hat Single Sign-On Operator 保护 Service Registry”](#)。

您可以使用基于使用 OpenID Connect (OIDC) 的 Red Hat Single Sign-On 的 Red Hat Single Sign-On 为 Service Registry web 控制台和核心 REST API 启用身份验证。相同的 Red Hat Single Sign-On 域和用户在使用 OpenID Connect 的 Service Registry web 控制台和核心 REST API 中联邦, 以便您只需要一组凭证。

Service Registry 为默认的 admin、write 和 read-only 用户角色提供基于角色的授权。Service Registry 在 schema 或 API 级别上提供基于内容的授权, 只有 registry 工件的创建者可以更新或删除它。Service Registry 身份验证和授权设置默认为禁用。

先决条件

- Red Hat Single Sign-On 已安装并运行。如需了解更多详细信息, 请参阅 [Red Hat Single Sign-On 用户文档](#)。
- Service Registry 已安装并运行。

流程

- 在 Red Hat Single Sign-On Admin 控制台中, 为 Service Registry 创建一个 Red Hat Single Sign-On 域。默认情况下, Service Registry 需要 registry 的域名。有关创建域的详情, 请查看 [Red Hat Single Sign-On 用户文档](#)。

2. 为 Service Registry API 创建 Red Hat Single Sign-On 客户端。默认情况下，Service Registry 需要以下设置：

- 客户端 ID : `registry-api`
- 客户端协议 : `openid-connect`
- 访问类型:`bearer-only`
您可以将默认值用于其他客户端设置。



注意

如果您使用 Red Hat Single Sign-On 服务帐户，客户端的 Access Type 必须是 `confidential` 而不是 `bearer-only`。

3. 为 Service Registry web 控制台创建 Red Hat Single Sign-On 客户端。默认情况下，Service Registry 需要以下设置：

- 客户端 ID:`apicurio-registry`
- 客户端协议 : `openid-connect`
- 访问类型 : `public`
- 有效的重定向 URL:`http://my-registry-url:8080/*`
- Web Origins:+
您可以将默认值用于其他客户端设置。

4. 在 OpenShift 上的 Service Registry 部署中，设置以下 Service Registry 环境变量以使用 Red Hat Single Sign-On 配置身份验证：

表 5.2. 使用 Red Hat Single Sign-On 配置 Service Registry 身份验证

环境变量	描述	类型	默认
AUTH_ENABLED	为 Service Registry 启用身份验证。当设置为 <code>true</code> 时，需要使用 Red Hat Single Sign-On 进行身份验证所需的环境变量。	字符串	false
KEYCLOAK_URL	Red Hat Single Sign-On 身份验证服务器的 URL。例如： http://localhost:8080 。	字符串	-
KEYCLOAK_REALM	用于身份验证的 Red Hat Single Sign-On 域。例如， registry 。	字符串	-
KEYCLOAK_API_CLIENT_ID	Service Registry REST API 的客户端 ID。	字符串	registry-api
KEYCLOAK_UI_CLIENT_ID	Service Registry web 控制台的客户端 ID。	字符串	apicurio-registry

提示

有关在 OpenShift 中设置环境变量的示例，请参阅 [第 6.1 节“在 OpenShift 中配置 Service Registry 健康检查”](#)。

- 将以下选项设置为 `true` 在 Red Hat Single Sign-On 中启用 Service Registry 用户角色：

表 5.3. 配置基于 Service Registry 角色的授权

环境变量	Java 系统属性	类型	默认值
<code>ROLE_BASED_AUTHZ_ENABLED</code>	<code>registry.auth.role-based-authorization</code>	布尔值	<code>false</code>

- 启用 Service Registry 用户角色时，您必须将 Service Registry 用户至少分配给 Red Hat Single Sign-On 域中的以下默认用户角色之一：

表 5.4. 用于 registry 身份验证和授权的默认用户角色

角色	读取工件	写入工件	全局规则	概述
<code>sr-admin</code>	是	是	是	所有创建、读取、更新和删除操作的完整访问权限。
<code>sr-developer</code>	是	是	否	除配置全局规则外，创建、读取、更新和删除操作的访问权限。此角色可配置特定于工件的规则。
<code>sr-readonly</code>	是	否	否	仅访问读和搜索操作。此角色无法配置任何规则。

- 将以下内容设置为 `true`，为 Service Registry 中的 schema 和 API 工件启用仅所有者授权：

表 5.5. 配置仅限所有者授权

环境变量	Java 系统属性	类型	默认值
<code>REGISTRY_AUTH_OBAC_ENABLED</code>	<code>registry.auth.owner-only-authorization</code>	布尔值	<code>false</code>

其他资源

- 有关配置非默认用户角色名称的详情，请参考 [第 5.3 节“Service Registry 身份验证和授权配置选项”](#)。
- 有关开源示例应用程序和 Keycloak 域，请参阅使用 Keycloak 的 [Apicurio Registry 的 Docker Compose 示例](#)。
- 有关如何在生产环境中使用 Red Hat Single Sign-On 的详情，请查看 [Red Hat Single Sign-On 文档](#)。

5.3. SERVICE REGISTRY 身份验证和授权配置选项

Service Registry 为 OpenID Connect 提供带有 Red Hat Single Sign-On 和 HTTP 基本身份验证的身份验证选项。

Service Registry 为基于角色的方法和基于内容的方法提供授权选项：

- 基于角色的默认授权，用于默认 admin、write 和 read-only 用户角色。
- 对于 schema 或 API 工件，基于内容的授权，只有工件或工件组的所有者才能更新或删除工件。



重要

Service Registry 中的所有身份验证和授权选项都默认禁用。在启用任何这些选项前，您必须首先将 `AUTH_ENABLED` 选项设置为 `true`。

本章详细介绍了以下配置选项：

- [使用带有红帽单点登录的 OpenID Connect 进行 Service Registry 身份验证](#)
- [使用 HTTP 基本的 Service Registry 身份验证](#)
- [基于 Service Registry 角色的授权](#)
- [仅 Service Registry 所有者授权](#)
- [Service Registry 验证的读访问权限](#)
- [Service Registry 匿名只读访问](#)

使用带有红帽单点登录的 OpenID Connect 进行 Service Registry 身份验证

您可以设置以下环境变量，以使用 Red Hat Single Sign-On 为 Service Registry web 控制台和 API 配置身份验证：

表 5.6. 使用 Red Hat Single Sign-On 配置 Service Registry 身份验证

环境变量	描述	类型	默认
<code>AUTH_ENABLED</code>	为 Service Registry 启用身份验证。当设置为 <code>true</code> 时，需要使用 Red Hat Single Sign-On 进行身份验证所需的环境变量。	字符串	<code>false</code>
<code>KEYCLOAK_URL</code>	Red Hat Single Sign-On 身份验证服务器的 URL。例如： <code>http://localhost:8080</code> 。	字符串	-
<code>KEYCLOAK_REALM</code>	用于身份验证的 Red Hat Single Sign-On 域。例如， <code>registry</code> 。	字符串	-
<code>KEYCLOAK_API_CLIENT_ID</code>	Service Registry REST API 的客户端 ID。	字符串	<code>registry-api</code>

环境变量	描述	类型	默认
KEYCLOAK_UI_CLIENT_ID	Service Registry web 控制台的客户端 ID。	字符串	apicurio-registry

使用 HTTP 基本的 Service Registry 身份验证

默认情况下，Service Registry 支持使用 OpenID Connect 进行身份验证。用户或 API 客户端必须获取访问令牌，才能对 Service Registry REST API 进行经过身份验证的调用。但是，由于某些工具不支持 OpenID Connect，因此您也可以将 Service Registry 配置为支持 HTTP 基本身份验证，方法是将以下配置选项设置为 **true**：

表 5.7. Service Registry HTTP 基本身份验证配置

环境变量	Java 系统属性	类型	默认值
AUTH_ENABLED	registry.auth.enabled	布尔值	false
CLIENT_CREDENTIALS_BASIC_AUTH_ENABLED	registry.auth.basic-auth-client-credentials.enabled	布尔值	false

Service Registry HTTP 基本客户端凭证缓存到期

您还可以配置 HTTP 基本客户端凭证缓存到期时间。默认情况下，在使用 HTTP 基本身份验证时，Service Registry 会缓存 JWT 令牌，并在不需要时不会发布新令牌。您可以为 JWT 令牌配置缓存到期时间，该令牌默认设置为 10 分钟。

使用 Red Hat Single Sign-On 时，最好将此配置设置为 Red Hat Single Sign-On JWT 到期时间减一分钟。例如，如果您在 Red Hat Single Sign-On 中将到期时间设置为 5 分钟，您应该将以下配置选项设置为 4 分钟：

表 5.8. 配置 HTTP 基本客户端凭证缓存到期

环境变量	Java 系统属性	类型	默认值
CLIENT_CREDENTIALS_BASIC_CACHE_EXPIRATION	registry.auth.basic-auth-client-credentials.cache-expiration	整数	10

基于 Service Registry 角色的授权

您可以将以下选项设置为 **true**，以便在 Service Registry 中启用基于角色的授权：

表 5.9. 配置基于 Service Registry 角色的授权

环境变量	Java 系统属性	类型	默认值
AUTH_ENABLED	registry.auth.enabled	布尔值	false
ROLE_BASED_AUTHZ_ENABLED	registry.auth.role-based-authorization	布尔值	false

然后，您可以将基于角色的授权配置为使用用户身份验证令牌中包含的角色（例如，在使用 Red Hat Single Sign-On 进行身份验证时授予），或者使用由 Service Registry 内部管理的角色映射。

使用 Red Hat Single Sign-On 中分配的角色

要使用 Red Hat Single Sign-On 分配的角色启用，请设置以下环境变量：

表 5.10. 使用红帽单点登录配置 Service Registry 基于角色的授权

环境变量	描述	类型	默认
ROLE_BASED_AUTHZ_SOURCE	当设置为 令牌 时，将从身份验证令牌获取用户角色。	字符串	token
REGISTRY_AUTH_ROLES_ADMIN	指明用户的角色名称是管理员。	字符串	sr-admin
REGISTRY_AUTH_ROLES_DEVELOPER	指明用户的角色名称是开发人员。	字符串	sr-developer
REGISTRY_AUTH_ROLES_READONLY	指明用户具有只读访问权限的角色名称。	字符串	sr-readonly

当 Service Registry 配置为使用 Red Hat Single Sign-On 中的角色时，您必须将 Service Registry 用户至少分配给 Red Hat Single Sign-On 中的以下用户角色之一。但是，您可以使用 [表 5.10 “使用红帽单点登录配置 Service Registry 基于角色的授权”](#) 中的环境变量配置不同的用户角色名称。

表 5.11. 用于身份验证和授权的 Service Registry 角色

角色名称	读取工件	写入工件	全局规则	描述
sr-admin	是	是	是	所有创建、读取、更新和删除操作的完整访问权限。
sr-developer	是	是	否	除配置全局规则和导入/导出外，创建、读取、更新和删除操作的访问权限。此角色只能配置特定于工件的规则。
sr-readonly	是	否	否	仅访问读和搜索操作。此角色无法配置任何规则。

在 Service Registry 中直接管理角色

要使用由 Service Registry 内部管理的角色启用，请设置以下环境变量：

表 5.12. 使用内部角色映射配置 Service Registry 基于角色的授权

环境变量	描述	类型	默认
ROLE_BASED_AUTHZ_SOURCE	当设置为 application 时，用户角色由 Service Registry 在内部管理。	字符串	token

在使用内部管理的角色映射时，可以使用 Service Registry REST API 中的 `/admin/roleMappings` 端点来为用户分配角色。如需了解更多详细信息，请参阅 [Apicurio Registry REST API 文档](#)。

用户可以精确授予一个角色：`ADMIN`、`DEVELOPER` 或 `READ_ONLY`。只有具有 `admin` 特权的用户才能向其他用户授予访问权限。

Service Registry admin-override 配置

因为 Service Registry 中没有默认的 `admin` 用户，因此通常会配置另一个用户被识别为 `admins` 的方法。您可以使用以下环境变量配置此 `admin-override` 功能：

表 5.13. Service Registry admin-override 的配置

环境变量	描述	类型	默认
<code>REGISTRY_AUTH_ADMIN_OVERRIDE_ENABLED</code>	启用 <code>admin-override</code> 功能。	字符串	<code>false</code>
<code>REGISTRY_AUTH_ADMIN_OVERRIDE_FROM</code>	在哪里查找 <code>admin-override</code> 信息。目前只支持 令牌 。	字符串	<code>token</code>
<code>REGISTRY_AUTH_ADMIN_OVERRIDE_TYPE</code>	用于确定用户是否是管理员的信息类型。值取决于 <code>FROM</code> 变量的值，例如当 <code>FROM</code> 为令牌时的 角色 或 声明 。	字符串	<code>role</code>
<code>REGISTRY_AUTH_ADMIN_OVERRIDE_ROLE</code>	指明用户的角色名称是管理员。	字符串	<code>sr-admin</code>
<code>REGISTRY_AUTH_ADMIN_OVERRIDE_CLAIM</code>	用于确定 <code>admin-override</code> 的 JWT 令牌声明的名称。	字符串	<code>org-admin</code>
<code>REGISTRY_AUTH_ADMIN_OVERRIDE_CLAIM_VALUE</code>	<code>CLAIM</code> 变量指示的 JWT 令牌声明值必须是被授予 <code>admin-override</code> 的用户。	字符串	<code>true</code>

例如，您可以使用此 `admin-override` 功能将 `sr-admin` 角色分配给 Red Hat Single Sign-On 中的单个用户，这将为该用户授予 `admin` 角色。然后，用户可以使用 `/admin/roleMappings` REST API（或关联的 UI）向其他用户授予角色（包括其他管理员）。

仅 Service Registry 所有者授权

您可以将以下选项设置为 `true`，为对 Service Registry 中的工件或工件组更新启用仅所有者授权：

表 5.14. 配置仅限所有者授权

环境变量	Java 系统属性	类型	默认值
<code>AUTH_ENABLED</code>	<code>registry.auth.enabled</code>	布尔值	<code>false</code>

环境变量	Java 系统属性	类型	默认值
REGISTRY_AUTH_OBAC_ENABLE_D	registry.auth.owner-only-authorization	布尔值	false
REGISTRY_AUTH_OBAC_LIMIT_GROUPS_ACCESS	registry.auth.owner-only-authorization.limit-group-access	布尔值	false

当启用了仅限所有者授权时，只有创建工件的用户才能修改或删除该工件。

当同时启用了所有者授权和组所有者授权时，只有创建工件组的用户才能对该工件组具有写入访问权限，例如要在该组中添加或删除工件。

Service Registry 验证的读访问权限

当启用经过身份验证的用户时，Service Registry 会至少授予来自同一机构中任何经过身份验证的用户的只读访问权限，而不考虑其用户角色。

要启用经过身份验证的读访问权限，您必须首先启用基于角色的授权，然后确保将以下选项设置为 **true**：

表 5.15. 配置经过身份验证的用户

环境变量	Java 系统属性	类型	默认值
AUTH_ENABLED	registry.auth.enabled	布尔值	false
REGISTRY_AUTH_AUTHENTICATED_READ_READS_ENABLED	registry.auth.authenticated-read-access.enabled	布尔值	false

如需了解更多详细信息，请参阅[“基于 Service Registry 角色的授权”](#)一节。

Service Registry 匿名只读访问

除了两种类型的授权（基于角色和基于所有者的授权），Service Registry 支持匿名只读访问选项。

要允许匿名用户（如没有身份验证凭证的 REST API 调用）对 REST API 进行只读调用，请将以下选项设置为 **true**：

表 5.16. 配置匿名只读访问

环境变量	Java 系统属性	类型	默认值
AUTH_ENABLED	registry.auth.enabled	布尔值	false
REGISTRY_AUTH_ANONYMOUS_READ_ACCESS_ENABLED	registry.auth.anonymous-read-access.enabled	布尔值	false

其他资源

- 有关如何在 OpenShift 上的 Service Registry 部署中设置环境变量的示例，请参阅[第 6.1 节“在 OpenShift 中配置 Service Registry 健康检查”](#)

- 有关为 Service Registry 配置自定义身份验证的详情，请参阅 [Quarkus Open ID Connect 文档](#)

5.4. 从 OPENSIFT 集群内部配置到 SERVICE REGISTRY 的 HTTPS 连接

以下流程演示了如何配置 Service Registry 部署，以便为来自 OpenShift 集群内的 HTTPS 连接公开端口。



警告

这种类型的连接在集群外不直接提供。路由基于主机名，该主机名在 HTTPS 连接的情况下是编码的。因此，仍然需要边缘终止或其他配置。请参阅 [第 5.5 节“从 OpenShift 集群外部配置到 Service Registry 的 HTTPS 连接”](#)。

先决条件

- 您必须已安装了 Service Registry Operator。

流程

1. 使用自签名证书生成 密钥存储。如果您使用自己的证书，可以跳过这一步。

```
openssl req -newkey rsa:2048 -new -nodes -x509 -days 3650 -keyout tls.key -out tls.crt
```

2. 创建新机密来保存证书和私钥。

- a. 在 OpenShift Web 控制台左侧导航菜单中，点 Workloads > Secrets > Create Key/Value Secret。

- b. 使用以下值：

Name: **https-cert-secret**

Key 1: **tls.key**

Value 1: *tls.key* (uploaded file)

Key 2: **tls.crt**

Value 2: *tls.crt* (uploaded file)

或者使用以下命令创建 secret：

```
oc create secret generic https-cert-secret --from-file=tls.key --from-file=tls.crt
```

3. 为您的 Service Registry 部署编辑 ApicurioRegistry CR 的 `spec.configuration.security.https` 部分，例如：

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
spec:
  configuration:
    # ...
```

```
security:
  https:
    secretName: https-cert-secret
```

4. 验证连接是否正常工作：

- a. 使用 SSH 连接到集群中的 pod（您可以使用 Service Registry pod）：

```
oc rsh example-apicurioregistry-deployment-6f788db977-2wzpw
```

- b. 从 Service 资源查找 Service Registry pod 的集群 IP（请参阅 web 控制台中的 Location 列）。之后，执行测试请求（我们使用自签名证书，因此需要一个 insecure 标志）：

```
curl -k https://172.30.230.78:8443/health
```



注意

在包含 HTTPS 证书和密钥的 Kubernetes secret 中，名称 `tls.crt` 和 `tls.key` 必须用于提供的值。这是目前不可配置。

禁用 HTTP

如果使用本节中的流程启用了 HTTPS，您也可以通过将 `spec.security.https.disableHttp` 设置为 `true` 来禁用默认的 HTTP 连接。这会从 Service Registry pod 容器、Service 和 NetworkPolicy（如果存在）中删除 HTTP 端口 8080。

重要的是，Ingress 也被删除，因为 Service Registry Operator 目前不支持在 Ingress 中配置 HTTPS。用户必须为 HTTPS 连接手动创建 Ingress。

其他资源

- [如何在 Quarkus 应用程序中启用 HTTPS 和 SSL 终止](#)

5.5. 从 OPENSIFT 集群外部配置到 SERVICE REGISTRY 的 HTTPS 连接

以下流程演示了如何配置 Service Registry 部署，以便为来自 OpenShift 集群外部的连接公开 HTTPS 边缘终止路由。

先决条件

- 您必须已安装了 Service Registry Operator。
- 阅读[用于创建安全路由的 OpenShift 文档](#)。

流程

1. 除了 Service Registry Operator 创建的 HTTP 路由外，添加第二个路由。例如：

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  [...]
labels:
  app: example-apicurioregistry
```

```
[...]  
spec:  
  host: example-apicuriregistry-default.apps.example.com  
  to:  
    kind: Service  
    name: example-apicuriregistry-service-9whd7  
    weight: 100  
  port:  
    targetPort: 8080  
  tls:  
    termination: edge  
    insecureEdgeTerminationPolicy: Redirect  
    wildcardPolicy: None
```



注意

确保设置了 `insecureEdgeTerminationPolicy: Redirect` 配置属性。

如果没有指定证书，OpenShift 将使用默认证书。另外，您可以使用以下命令生成自定义自签名证书：

```
openssl genrsa 2048 > tls.key &&  
openssl req -new -x509 -nodes -sha256 -days 365 -key tls.key -out tls.crt
```

然后，使用 OpenShift CLI 创建路由：

```
oc create route edge \  
  --service=example-apicuriregistry-service-9whd7 \  
  --cert=tls.crt --key=tls.key \  
  --hostname=example-apicuriregistry-default.apps.example.com \  
  --insecure-policy=Redirect \  
  -n default
```

第 6 章 配置和管理 SERVICE REGISTRY 部署

本章介绍了如何在 OpenShift 上为 Service Registry 部署配置和管理可选设置：

- [第 6.1 节 “在 OpenShift 中配置 Service Registry 健康检查”](#)
- [第 6.2 节 “Service Registry 健康检查的环境变量”](#)
- [第 6.3 节 “管理 Service Registry 环境变量”](#)
- [第 6.4 节 “使用 PodTemplate 配置 Service Registry 部署”](#)
- [第 6.5 节 “配置 Service Registry web 控制台”](#)
- [第 6.6 节 “配置 Service Registry 日志”](#)
- [第 6.7 节 “配置 Service Registry 事件源”](#)

6.1. 在 OPENSIFT 中配置 SERVICE REGISTRY 健康检查

您可以为存活度和就绪度探测配置可选环境变量，以监控 OpenShift 上的 Service Registry 服务器的健康状态：

- **存活度探测** 测试应用是否可以进行进度。如果应用无法进行，OpenShift 会自动重启失败的 Pod。
- **就绪度探测** 测试应用是否准备好处理请求。如果应用未就绪，则请求可能会变得非常严重，OpenShift 会停止在探测失败时发送请求。如果其他 Pod 是 OK，它们将继续接收请求。



重要

存活度和就绪度环境变量的默认值是为大多数情况设计的，只有在您的环境需要时才应更改。对默认值的任何更改取决于您的硬件、网络和存储的数据量。这些值应尽可能低，以避免不必要的开销。

先决条件

- 您必须具有具有集群管理员访问权限的 OpenShift 集群。
- 您必须已在 OpenShift 上安装 Service Registry。
- 您必须已在 AMQ Streams 或 PostgreSQL 中安装并配置了所选 Service Registry 存储。

流程

1. 在 OpenShift Container Platform Web 控制台中，使用具有集群管理员特权的帐户登录。
2. 点 Installed Operators > Red Hat Integration - Service Registry Operator
3. 在 ApicurioRegistry 选项卡中，点部署的 Operator 自定义资源，如 example-apicurioregistry。
4. 在主概览页面中，找到 Service Registry 部署的 Deployment Name 部分和对应的 DeploymentConfig 名称，如 example-apicurioregistry。

5. 在左侧导航菜单中，点 Workloads > Deployment Configs，然后选择您的 DeploymentConfig 名称。
6. 点 Environment 选项卡，然后在 Single values env 部分中输入环境变量，例如：
 - 名称：LIVENESS_STATUS_RESET
 - 值：350
7. 点底部的 Save。
或者，您可以使用 OpenShift oc 命令执行这些步骤。如需了解更多详细信息，请参阅 [OpenShift CLI 文档](#)。

其他资源

- [第 6.2 节 “Service Registry 健康检查的环境变量”](#)
- [OpenShift 文档中有关监控应用程序健康状况的文档](#)

6.2. SERVICE REGISTRY 健康检查的环境变量

本节介绍 OpenShift 上 Service Registry 健康检查的可用环境变量。这包括存活度和就绪度探测，以监控 OpenShift 上 Service Registry 服务器的健康状态。有关示例步骤，请参阅 [第 6.1 节 “在 OpenShift 中配置 Service Registry 健康检查”](#)。



重要

提供的以下环境变量仅供参考。默认值是为大多数情况设计的，只有在您的环境需要时才应更改。对默认值的任何更改取决于您的硬件、网络和存储的数据量。这些值应尽可能低，以避免不必要的开销。

存活度环境变量

表 6.1. Service Registry 存活度探测的环境变量

名称	描述	类型	默认
LIVENESS_ERROR_THRESHOLD	存活度探测失败前可能会出现存活度问题或错误的数量。	整数	1
LIVENESS_COUNTER_RESET	必须发生错误的阈值数的期间。例如，如果这个值为 60，阈值为 1，则检查在 1 分钟内出现两个错误后会失败	秒	60
LIVENESS_STATUS_RESET	存活度探测的无错误要经过的秒数才能重置为 OK 状态。	秒	300
LIVENESS_ERRORS_IGNORED	以逗号分隔的忽略存活度例外列表。	字符串	io.grpc.StatusRuntimeException,org.apache.kafka.streams.errors.InvalidStateStoreException



注意

由于 OpenShift 会自动重启一个无法存活度检查的 Pod，所以存活度设置与就绪度设置不同，不会影响 OpenShift 上 Service Registry 的行为。

就绪度环境变量

表 6.2. Service Registry 就绪度探测的环境变量

名称	描述	类型	默认
READINESS_ERROR_THRESHOLD	就绪度探测失败前可能出现的就绪度问题或错误数量。	整数	1
READINESS_COUNTER_RESET	必须发生错误的阈值数的期间。例如，如果这个值为 60，阈值为 1，则检查在 1 分钟内出现两个错误后会失败。	秒	60
READINESS_STATUS_RESET	存活度探测的无错误要经过的秒数才能重置为 OK 状态。在这种情况下，这意味着 Pod 保持未就绪的时间，直到它返回到正常操作为止。	秒	300
READINESS_TIMEOUT	readiness 跟踪两个操作的超时时间： <ul style="list-style-type: none"> • 存储请求完成所需的时间 • HTTP REST API 请求返回响应所需的时间 如果这些操作的时间超过配置的超时时间，这将计为就绪度问题或错误。这个值控制这两个操作的超时时间。	秒	5

其他资源

- [第 6.1 节 “在 OpenShift 中配置 Service Registry 健康检查”](#)
- [OpenShift 文档中有关监控应用程序健康状况的文档](#)

6.3. 管理 SERVICE REGISTRY 环境变量

Service Registry Operator 管理最常见的 Service Registry 配置，但有一些选项还不支持它。如果 ApicurioRegistry CR 中没有高级别配置选项，您可以使用环境变量来调整它。您可以通过在 `spec.configuration.env` 字段中直接设置 ApicurioRegistry CR 中的环境变量来更新它们。然后，它们转发到 Service Registry 的 Deployment 资源。

流程

您可以使用 OpenShift Web 控制台或 CLI 管理 Service Registry 环境变量。

OpenShift web 控制台

1. 选择 Installed Operators 选项卡，然后选择 Red Hat Integration - Service Registry Operator。
2. 在 Apicurio Registry 标签页中，为您的 Service Registry 部署选择 ApicurioRegistry CR。
3. 点 YAML 选项卡，然后根据需要编辑 `spec.configuration.env` 部分。以下示例演示了如何设置默认全局内容规则：

```

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
spec:
  configuration:
    # ...
    env:
      - name: REGISTRY_RULES_GLOBAL_VALIDITY
        value: FULL # One of: NONE, SYNTAX_ONLY, FULL
      - name: REGISTRY_RULES_GLOBAL_COMPATIBILITY
        value: FULL # One of: NONE, BACKWARD, BACKWARD_TRANSITIVE,
FORWARD, FORWARD_TRANSITIVE, FULL, FULL_TRANSITIVE

```

OpenShift CLI

1. 选择安装 Service Registry 的项目。
2. 运行 `oc get apicurioregistry` 以获取 ApicurioRegistry CR 列表
3. 在代表您要配置的 Service Registry 实例的 CR 上运行 `oc edit apicurioregistry`。
4. 在 `spec.configuration.env` 部分添加或修改环境变量。
Service Registry Operator 可能会尝试设置已在 `spec.configuration.env` 字段中明确指定的环境变量。如果环境变量配置具有冲突的值，则 Service Registry Operator 设置的值具有优先权。

您可以通过对该功能使用高级别配置或仅使用明确指定的环境变量来避免这种冲突。以下是冲突配置示例：

```

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
spec:
  configuration:
    # ...
    ui:
      readOnly: true
    env:
      - name: REGISTRY_UI_FEATURES_READONLY
        value: false

```

此配置会导致 Service Registry web 控制台处于只读模式。

6.4. 使用 PODTEMPLATE 配置 SERVICE REGISTRY 部署



重要

这只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。

这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅

<https://access.redhat.com/support/offerings/techpreview>。

ApicurioRegistry CRD 包含 `spec.deployment.podTemplateSpecPreview` 字段，它的结构与 Kubernetes `Deployment` 资源(`PodTemplateSpec` struct)中的 `spec.template` 字段相同。

通过一些限制，Service Registry Operator 将来自此字段的数据转发到 Service Registry 部署中的对应字段。这提供了更大的灵活性，而无需 Service Registry Operator 才能原生支持每个用例。

下表包含 Service Registry Operator 不接受的子字段列表，并导致配置错误：

表 6.3. `podTemplateSpecPreview` 子字段的限制

podTemplateSpecPreview subfield	Status	详情
<code>metadata.annotations</code>	其它方法存在	<code>spec.deployment.metadata.annotations</code>
<code>metadata.labels</code>	其它方法存在	<code>spec.deployment.metadata.labels</code>
<code>spec.affinity</code>	其它方法存在	<code>spec.deployment.affinity</code>
<code>spec.containers[*]</code>	warning	要配置 Service Registry 容器，必须使用 <code>name: registry</code>
<code>spec.containers[name = "registry"].env</code>	其它方法存在	<code>spec.configuration.env</code>
<code>spec.containers[name = "registry"].image</code>	保留	-
<code>spec.imagePullSecrets</code>	其它方法存在	<code>spec.deployment.imagePullSecrets</code>
<code>spec.tolerations</code>	其它方法存在	<code>spec.deployment.tolerations</code>



警告

如果在 `podTemplateSpecPreview` 中设置字段，则其值必须有效，就像直接在 `Service Registry Deployment` 中配置一样。Service Registry Operator 可能仍然修改您提供的值，但不会修复无效的值或确保存在默认值。

其他资源

- [Kubernetes 文档中有关 Pod 模板的文档](#)

6.5. 配置 SERVICE REGISTRY WEB 控制台

您可以设置可选环境变量来配置 Service Registry web 控制台，专门用于部署环境或自定义其行为。

先决条件

- 已安装 Service Registry。

配置 Web 控制台部署环境

当您在浏览器中访问 Service Registry web 控制台时，会加载一些初始配置设置。以下配置设置非常重要：

- 核心 Service Registry 服务器 REST API 的 URL
- Service Registry Web 控制台客户端的 URL

通常，Service Registry 会自动检测并生成这些设置，但有些部署环境可能会失败。如果发生这种情况，您可以配置环境变量以明确为您的环境设置这些 URL。

流程

配置以下环境变量以覆盖默认 URL：

- `REGISTRY_UI_CONFIG_APIURL`：指定核心 Service Registry 服务器 REST API 的 URL。例如：`https://registry.my-domain.com/apis/registry`
- `REGISTRY_UI_CONFIG_UIURL`：指定 Service Registry Web 控制台客户端的 URL。例如：`https://registry.my-domain.com/ui`

以只读模式配置 Web 控制台

您可以使用只读模式将 Service Registry web 控制台配置为可选功能。此模式禁用 Service Registry web 控制台中的所有功能，允许用户更改注册的工件。例如，这包括以下内容：

- 创建工件
- 上传新的工件版本
- 更新工件元数据
- 删除工件

流程

配置以下环境变量：

- `REGISTRY_UI_FEATURES_READONLY`: 设置为 `true` 以启用只读模式。默认值为 `false`。

6.6. 配置 SERVICE REGISTRY 日志

您可以在运行时设置 Service Registry 日志记录配置。Service Registry 提供了一个 REST 端点，用于为特定日志记录器设置日志级别，以进行精细的日志记录。本节介绍如何使用 Service Registry `/admin` REST API 在运行时查看和设置 Service Registry 日志级别。

先决条件

- 获取用于访问 Service Registry 实例的 URL，如果 OpenShift 上部署了 Service Registry，则获取您的 Service Registry 路由。此简单示例使用 `localhost:8080` 的 URL。

流程

1. 使用此 `curl` 命令获取日志记录器 `io.apicurio.registry.storage` 的当前日志级别：

```
$ curl -i localhost:8080/apis/registry/v2/admin/loggers/io.apicurio.registry.storage
HTTP/1.1 200 OK
[...]
Content-Type: application/json
{"name":"io.apicurio.registry.storage","level":"INFO"}
```

2. 使用此 `curl` 命令将日志记录器 `io.apicurio.registry.storage` 的日志级别更改为 `DEBUG`：

```
$ curl -X PUT -i -H "Content-Type: application/json" --data '{"level":"DEBUG"}'
localhost:8080/apis/registry/v2/admin/loggers/io.apicurio.registry.storage
HTTP/1.1 200 OK
[...]
Content-Type: application/json
{"name":"io.apicurio.registry.storage","level":"DEBUG"}
```

3. 使用此 `curl` 命令将日志记录器 `io.apicurio.registry.storage` 的日志级别恢复到其默认值：

```
$ curl -X DELETE -i
localhost:8080/apis/registry/v2/admin/loggers/io.apicurio.registry.storage
HTTP/1.1 200 OK
[...]
Content-Type: application/json
{"name":"io.apicurio.registry.storage","level":"INFO"}
```

6.7. 配置 SERVICE REGISTRY 事件源



重要

这只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。

这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅

<https://access.redhat.com/support/offerings/techpreview>。

您可以将 Service Registry 配置为在更改 registry 内容时发送事件。例如，当创建 schema 或 API 工件、组或内容规则时，Service Registry 可以在创建、更新、删除等时触发事件。您可以将 Service Registry 配置为向应用程序发送事件，并将这些更改发送到第三方集成。

有不同的协议可用于传输事件。目前实现的协议是 HTTP 和 Apache Kafka。但是，无论协议如何，使用 CNCF CloudEvents 规格发送事件。您可以使用 Java 系统属性或等同的环境变量配置 Service Registry 事件源。

Service Registry 事件类型

所有事件类型都在 `io.apicurio.registry.events.dto.RegistryEventType` 中定义。例如，它们包括以下事件类型：

- `io.apicurio.registry.artifact-created`
- `io.apicurio.registry.artifact-updated`
- `io.apicurio.registry.artifact-state-changed`
- `io.apicurio.registry.artifact-rule-created`
- `io.apicurio.registry.global-rule-created`
- `io.apicurio.registry.group-created`

先决条件

- 您必须具有要将 Service Registry 云事件发送到的应用程序。例如，这可能是自定义应用程序或第三方应用程序。

使用 HTTP 配置 Service Registry 事件源

本节中的示例显示了在 `http://my-app-host:8888/events` 上运行的自定义应用程序。

流程

1. 使用 HTTP 协议时，将 Service Registry 配置设置为将事件发送到应用程序，如下所示：
 - `registry.events.sink.my-custom-consumer=http://my-app-host:8888/events`
2. 如果需要，您可以按照以下方式配置多个事件消费者：
 - `registry.events.sink.my-custom-consumer=http://my-app-host:8888/events`
 - `registry.events.sink.other-consumer=http://my-consumer.com/events`

使用 Apache Kafka 配置 Service Registry 事件源

本节中的示例显示名为 `my-registry-events` 的 Kafka 主题，在 `my-kafka-host:9092` 上运行。

流程

1. 使用 Kafka 协议时，按如下所示设置 Kafka 主题：
 - `registry.events.kafka.topic=my-registry-events`
2. 您可以使用 `KAFKA_BOOTSTRAP_SERVERS` 环境变量设置 Kafka producer 的配置：
 - `KAFKA_BOOTSTRAP_SERVERS=my-kafka-host:9092`

另外，您可以使用 `registry.events.kafka.config` 前缀设置 kafka producer 的属性，例如：
`registry.events.kafka.config.bootstrap.servers=my-kafka-host:9092`

3. 如果需要，您还可以设置 Kafka 主题分区来生成事件：

- `registry.events.kafka.topic-partition=1`

其他资源

- 如需了解更多详细信息，请参阅 [CNCF CloudEvents 规格](#)。

第 7 章 SERVICE REGISTRY OPERATOR 配置参考

本章详细介绍了用于配置 Service Registry Operator 以部署 Service Registry Operator 的自定义资源：

- [第 7.1 节 “Service Registry 自定义资源”](#)
- [第 7.2 节 “Service Registry CR spec”](#)
- [第 7.3 节 “Service Registry CR 状态”](#)
- [第 7.4 节 “Service Registry 受管资源”](#)
- [第 7.5 节 “Service Registry Operator 标签”](#)

7.1. SERVICE REGISTRY 自定义资源

Service Registry Operator 定义了一个 **ApicurioRegistry 自定义资源(CR)**，它代表 OpenShift 上的单个 Service Registry 部署。

这些资源对象由用户创建和维护，以指示 Service Registry Operator 如何部署和配置 Service Registry。

ApicurioRegistry CR 示例

以下命令显示 ApicurioRegistry 资源：

```
oc get apicurioregistry
oc edit apicurioregistry example-apicurioregistry

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
  namespace: demo-kafka
  # ...
spec:
  configuration:
    persistence: kafkasql
    kafkasql:
      bootstrapServers: 'my-cluster-kafka-bootstrap.demo-kafka.svc:9092'
  deployment:
    host: >-
      example-apicurioregistry.demo-kafka.example.com
status:
  conditions:
  - lastTransitionTime: "2021-05-03T10:47:11Z"
    message: ""
    reason: Reconciled
    status: "True"
    type: Ready
  info:
    host: example-apicurioregistry.demo-kafka.example.com
  managedResources:
  - kind: Deployment
    name: example-apicurioregistry-deployment
```

```

namespace: demo-kafka
- kind: Service
  name: example-apicurioregistry-service
namespace: demo-kafka
- kind: Ingress
  name: example-apicurioregistry-ingress
namespace: demo-kafka

```



重要

默认情况下，Service Registry Operator 只监视自己的项目命名空间。因此，如果要手动部署 Operator，则必须在同一命名空间中创建 `ApicurioRegistry` CR。您可以通过更新 Operator `Deployment` 资源中的 `WATCH_NAMESPACE` 环境变量来修改此行为。

其他资源

- [使用自定义资源定义来扩展 Kubernetes API](#)

7.2. SERVICE REGISTRY CR SPEC

`spec` 是 `ApicurioRegistry` CR 的一部分，用于为 Operator 提供所需状态或配置。

ApicurioRegistry CR spec 内容

以下示例块包含可能 `spec` 配置选项的完整树。有些字段可能不是必需字段，或者不应同时定义。

```

spec:
  configuration:
    persistence: <string>
    sql:
      dataSource:
        url: <string>
        userName: <string>
        password: <string>
      kafkasql:
        bootstrapServers: <string>
      security:
        tls:
          truststoreSecretName: <string>
          keystoreSecretName: <string>
        scram:
          mechanism: <string>
          truststoreSecretName: <string>
          user: <string>
          passwordSecretName: <string>
    ui:
      readOnly: <string>
    logLevel: <string>
    registryLogLevel: <string>
    security:
      keycloak:
        url: <string>
        realm: <string>
        apiClientId: <string>
        uiClientId: <string>

```

```

https:
  disableHttp: <bool>
  secretName: <string>
env: <k8s.io/api/core/v1 []EnvVar>
deployment:
  replicas: <int32>
  host: <string>
  affinity: <k8s.io/api/core/v1 Affinity>
  tolerations: <k8s.io/api/core/v1 []Toleration>
  imagePullSecrets: <k8s.io/api/core/v1 []LocalObjectReference>
  metadata:
    annotations: <map[string]string>
    labels: <map[string]string>
  managedResources:
    disableIngress: <bool>
    disableNetworkPolicy: <bool>
  disablePodDisruptionBudget: <bool>
  podTemplateSpecPreview: <k8s.io/api/core/v1 PodTemplateSpec>

```

下表描述了每个配置选项：

表 7.1. ApicurioRegistry CR spec 配置选项

配置选项	type	默认值	描述
配置	-	-	Service Registry 应用程序配置部分
configuration/persistence	字符串	<i>required</i>	存储后端。 sql,kafkasql 之一
configuration/sql	-	-	SQL 存储后端配置
configuration/sql/dataSource	-	-	SQL 存储后端的数据库连接配置
configuration/sql/dataSource/url	字符串	<i>required</i>	数据库连接 URL 字符串
configuration/sql/dataSource/username	字符串	<i>required</i>	数据库连接用户
configuration/sql/dataSource/password	字符串	<i>empty</i>	数据库连接密码
configuration/kafkasql	-	-	Kafka 存储后端配置
configuration/kafkasql/bootstrapServers	字符串	<i>required</i>	Kafka bootstrap 服务器 URL，用于流存储后端
configuration/kafkasql/security/tls	-	-	为 Kafka 存储后端配置 TLS 身份验证的部分

配置选项	type	默认值	描述
configuration/kafkasql/security/tls/truststoreSecretName	字符串	<i>required</i>	包含 Kafka 的 TLS 信任存储的 secret 名称
configuration/kafkasql/security/tls/keystoreSecretName	字符串	<i>required</i>	包含用户 TLS 密钥存储的 secret 名称
configuration/kafkasql/security/scram/truststoreSecretName	字符串	<i>required</i>	包含 Kafka 的 TLS 信任存储的 secret 名称
configuration/kafkasql/security/scram/user	字符串	<i>required</i>	SCRAM 用户名
configuration/kafkasql/security/scram/passwordSecretName	字符串	<i>required</i>	包含 SCRAM 用户密码的 secret 名称
configuration/kafkasql/security/scram/mechanism	字符串	SCRAM-SHA-512	SASL 机制
configuration/ui	-	-	Service Registry web 控制台设置
configuration/ui/readOnly	字符串	false	将 Service Registry web 控制台设置为只读模式
configuration/logLevel	字符串	INFO	Service Registry 日志级别，用于非Apicurio 组件和库。 INFO,DEBUG 之一
configuration/registryLogLevel	字符串	INFO	Service Registry 日志级别，用于 Apicurio 应用程序组件（不包括非Apicurio 组件和库）。 INFO,DEBUG 之一
configuration/security	-	-	Service Registry web 控制台和 REST API 安全设置
configuration/security/keycloak	-	-	使用红帽单点登录的 Web 控制台和 REST API 安全配置
configuration/security/keycloak/url	字符串	<i>required</i>	Red Hat Single Sign-On URL
configuration/security/keycloak/realm	字符串	<i>required</i>	Red Hat Single Sign-On 域

配置选项	type	默认值	描述
configuration/security/keycloak/apiClientId	字符串	registry-client-api	用于 REST API 的 Red Hat Single Sign-On 客户端
configuration/security/keycloak/uiClientId	字符串	registry-client-ui	Red Hat Single Sign-On 客户端用于 Web 控制台
configuration/security/https	-	-	配置 HTTPS。如需了解更多详细信息，请参阅从 OpenShift 集群内配置到 Service Registry 的 HTTPS 连接 。
configuration/security/https/secretName	字符串	<i>empty</i>	包含 HTTPS 证书和密钥的 Kubernetes Secret 名称，必须分别命名为 tls.crt 和 tls.key 。设置此字段可启用 HTTPS，反之亦然。
configuration/security/https/disableHttp	bool	false	禁用 HTTP 端口和 Ingress。HTTPS 必须启用为前提条件。
configuration/env	k8s.io/api/core/v1 []EnvVar	<i>empty</i>	配置要提供给 Service Registry pod 的环境变量列表。如需了解更多详细信息，请参阅 管理 Service Registry 环境变量 。
部署	-	-	Service Registry 部署设置部分
deployment/replicas	正整数	1	要部署的 Service Registry pod 数量
deployment/host	字符串	<i>自动生成的</i>	提供 Service Registry 控制台和 API 的主机/URL。如果可能，Service Registry Operator 会尝试根据集群路由器的设置来确定正确的值。该值仅自动生成一次，因此用户之后可以覆盖它。
deployment/affinity	k8s.io/api/core/v1 Affinity	<i>empty</i>	Service Registry 部署关联性配置
deployment/tolerations	k8s.io/api/core/v1 []Toleration	<i>empty</i>	Service Registry 部署容限配置

配置选项	type	默认值	描述
deployment/imagePullSecrets	k8s.io/api/core/v1 []LocalObjectReference	<i>empty</i>	为 Service Registry 部署配置镜像 pull secret
deployment/metadata	-	-	为 Service Registry pod 配置一组标签或注解。
deployment/metadata/labels	map[string]string	<i>empty</i>	为 Service Registry pod 配置一组标签
deployment/metadata/annotations	map[string]string	<i>empty</i>	为 Service Registry pod 配置一组注解
deployment/managedResources	-	-	配置 Service Registry Operator 如何管理 Kubernetes 资源的部分。如需了解更多详细信息，请参阅 Service Registry 受管资源 。
deployment/managedResources/disableIngress	bool	false	如果设置，Operator 将不会为 Service Registry 部署创建和管理 Ingress 资源。
deployment/managedResources/disableNetworkPolicy	bool	false	如果设置，Operator 将不会为 Service Registry 部署创建和管理 NetworkPolicy 资源。
deployment/managedResources/disablePodDisruptionBudget	bool	false	如果设置，Operator 将不会为 Service Registry 部署创建和管理 PodDisruptionBudget 资源。
deployment/podTemplateSpecPreview	k8s.io/api/core/v1 PodTemplateSpec	<i>empty</i>	配置 Service Registry 部署资源的部分。如需了解更多详细信息，请参阅使用 PodTemplate 配置 Service Registry 部署 。



注意

如果将某个选项标记为 **必需的**，则可能对正在启用的其他配置选项具有条件。可以接受空值，但 Operator 不执行指定的操作。

7.3. SERVICE REGISTRY CR 状态

status 是 Service Registry Operator 管理的 CR 部分，其中包含当前部署和应用程序状态的描述。

ApicurioRegistry CR 状态内容

status 部分包含以下字段：

```
status:
  info:
    host: <string>
  conditions: <list of:>
  - type: <string>
    status: <string, one of: True, False, Unknown>
    reason: <string>
    message: <string>
    lastTransitionTime: <string, RFC-3339 timestamp>
  managedResources: <list of:>
  - kind: <string>
    namespace: <string>
    name: <string>
```

表 7.2. ApicurioRegistry CR status 字段

status 字段	类型	描述
info	-	包含关于部署的 Service Registry 的信息的部分。
info/host	字符串	可以访问 Service Registry UI 和 REST API 的 URL。
conditions	-	报告 Service Registry 状态的条件列表，或与该部署相关的 Operator。
conditions/type	字符串	条件的类型。
conditions/status	字符串	条件的状态， True, False, Unknown 之一。
conditions/reason	字符串	程序标识符表示条件最后一次转换的原因。
conditions/message	字符串	人类可读的消息，指示关于转换的详细信息。
conditions/lastTransitionTime	字符串	条件从一个状态转换到另一个状态最后一次的时间。
managedResources	-	由 Service Registry Operator 管理的 OpenShift 资源列表
managedResources/kind	字符串	资源类型。
managedResources/namespace	字符串	资源命名空间。

status 字段	类型	描述
managedResources/name	字符串	资源名称。

7.4. SERVICE REGISTRY 受管资源

在部署 Service Registry 时，由 Service Registry Operator 管理的资源如下：

- Deployment
- Ingress（和路由）
- NetworkPolicy
- PodDisruptionBudget
- 服务

您可以禁用 Service Registry Operator 来创建和管理某些资源，以便手动配置它们。这在使用 Service Registry Operator 目前不支持的功能时提供更大的灵活性。

如果您禁用资源类型，则其现有实例将被删除。如果启用资源，Service Registry Operator 会尝试使用 `app` 标签查找资源，如 `app=example-apicurioregistry`，并开始管理它。否则，Operator 会创建一个新实例。

您可以以这种方式禁用以下资源类型：

- Ingress（和路由）
- NetworkPolicy
- PodDisruptionBudget

例如：

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
spec:
  deployment:
    managedResources:
      disableIngress: true
      disableNetworkPolicy: true
      disablePodDisruptionBudget: false # Can be omitted
```

7.5. SERVICE REGISTRY OPERATOR 标签

由 Service Registry Operator 管理的资源通常被标记为如下：

表 7.3. 受管资源的 Service Registry Operator 标签

标签	描述
app	根据指定 ApicurioRegistry CR 的名称，资源所属的 Service Registry 部署的名称。
apicur.io/type	部署类型： apicurio-registry 或 operator
apicur.io/name	部署的名称：与 app 或 apicurio-registry-operator 相同的值
apicur.io/version	Service Registry 或 Service Registry Operator 的版本
app.kubernetes.io/*	应用程序部署的一组推荐的 Kubernetes 标签。
com.company and rht.*	红帽产品的 metering 标签。

自定义标签和注解

您可以使用 `spec.deployment.metadata.labels` 和 `spec.deployment.metadata.annotations` 字段为 Service Registry pod 提供自定义标签和注解，例如：

```

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
spec:
  configuration:
    # ...
  deployment:
    metadata:
      labels:
        example.com/environment: staging
      annotations:
        example.com/owner: my-team

```

其他资源

- [为应用程序部署推荐的 Kubernetes 标签](#)

第 8 章 SERVICE REGISTRY 配置参考

本章提供了可用于 Service Registry 的配置选项的参考信息。

- [第 8.1 节 “Service Registry 配置选项”](#)

其他资源

- 有关使用 Core Registry API 设置配置选项的详情，请查看 [Apicurio Registry REST API 文档中的 /admin/config/properties 端点](#)。
- 有关 Kafka serializers 和反序列化器的客户端配置选项的详情，请参阅 [Service Registry 用户指南](#)。

8.1. SERVICE REGISTRY 配置选项

每个组件类别都有以下 Service Registry 配置选项：

8.1.1. api

表 8.1. API 配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
<code>registry.api.errors.include-stack-in-response</code>	布尔值	<code>false</code>	2.1.4.Final	在错误响应中包含堆栈追踪
<code>registry.disable.apis</code>	可选 <list<string>>		2.0.0.Final	禁用 API

8.1.2. auth

表 8.2. 身份验证配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
<code>registry.auth.admin-override.claim</code>	string	<code>org-admin</code>	2.1.0.Final	身份验证管理员覆盖声明
<code>registry.auth.admin-override.claim-value</code>	string	<code>true</code>	2.1.0.Final	身份验证 admin 覆盖声明值
<code>registry.auth.admin-override.enabled</code>	布尔值	<code>false</code>	2.1.0.Final	启用 auth admin 覆盖
<code>registry.auth.admin-override.from</code>	string	<code>token</code>	2.1.0.Final	Auth admin 覆盖 from

Name	类型	Default (默认)	可从以下位置获得	描述
registry.auth.admin-override.role	string	sr-admin	2.1.0.Final	身份验证 admin 覆盖角色
registry.auth.admin-override.type	string	role	2.1.0.Final	身份验证管理员覆盖类型
registry.auth.anonymous-read-access.enabled	boolean [dynamic]	false	2.1.0.Final	匿名读取访问
registry.auth.audit.log.prefix	string	audit	2.2.6	用于应用程序审计日志记录的前缀。
registry.auth.authenticated-read-access.enabled	boolean [dynamic]	false	2.1.4.Final	经过身份验证的读访问权限
registry.auth.basic-auth-client-credentials.cache-expiration	整数	10	2.2.6.Final	客户端凭据令牌过期时间。
registry.auth.basic-auth-client-credentials.enabled	boolean [dynamic]	false	2.1.0.Final	启用基本身份验证客户端凭证
registry.auth.basic-auth.scope	可选 <string>		2.5.0.Final	客户端凭据范围。
registry.auth.client-id	string		2.0.0.Final	服务器用来进行身份验证的客户端标识符。
registry.auth.client-secret	可选 <string>		2.1.0.Final	服务器用来进行身份验证的客户端机密。
registry.auth.enabled	布尔值	false	2.0.0.Final	启用身份验证
registry.auth.owner-only-authorization	boolean [dynamic]	false	2.0.0.Final	工件仅所有者授权
registry.auth.owner-only-authorization.limit-group-access	boolean [dynamic]	false	2.1.0.Final	工件组所有者-仅限授权
registry.auth.role-based-authorization	布尔值	false	2.1.0.Final	启用基于角色的授权
registry.auth.role-source	string	token	2.1.0.Final	身份验证角色源

Name	类型	Default (默认)	可从以下位置获得	描述
<code>registry.auth.role-source.header.name</code>	string		2.4.3.Final	标头授权名称
<code>registry.auth.roles.admin</code>	string	sr-admin	2.0.0.Final	身份验证角色 admin
<code>registry.auth.roles.developer</code>	string	sr-developer	2.1.0.Final	身份验证角色开发人员
<code>registry.auth.roles.readonly</code>	string	sr-readonly	2.1.0.Final	身份验证角色为只读
<code>registry.auth.tenant-owner-is-admin.enabled</code>	布尔值	true	2.1.0.Final	启用 auth 租户所有者 admin
<code>registry.auth.token.endpoint</code>	string		2.1.0.Final	身份验证服务器 url.

8.1.3. 缓存

表 8.3. 缓存配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
<code>registry.config.cache.enabled</code>	布尔值	true	2.2.2.Final	启用了 registry 缓存

8.1.4. ccompat

表 8.4. ccompat 配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
<code>registry.ccompat.legacy-id-mode.enabled</code>	boolean [dynamic]	false	2.0.2.Final	旧 ID 模式 (兼容性 API)
<code>registry.ccompat.max-subjects</code>	integer [dynamic]	1000	2.4.2.Final	返回的最大对象数 (兼容性 API)
<code>registry.ccompat.use-canonical-hash</code>	boolean [dynamic]	false	2.3.0.Final	规范哈希模式 (兼容性 API)

8.1.5. 下载

表 8.5. 下载配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
<code>registry.download.href.ttl</code>	long [dynamic]	30	2.1.2.Final	下载链接到期

8.1.6. Events

表 8.6. 事件配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
<code>registry.events.ksink</code>	可选 <string>		2.0.0.Final	启用事件 Kafka sink

8.1.7. Health

表 8.7. 健康配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
<code>registry.liveness.errors.ignored</code>	可选 <list<string>>		1.2.3.Final	忽略存活度错误
<code>registry.metrics.PersistenceExceptionLivenessCheck.counterResetWindowDurationSec</code>	整数	60	1.0.2.Final	持久性存活度检查的计数重置窗口持续时间
<code>registry.metrics.PersistenceExceptionLivenessCheck.disableLogging</code>	布尔值	false	2.0.0.Final	禁用持久性存活度检查的日志记录
<code>registry.metrics.PersistenceExceptionLivenessCheck.errorThreshold</code>	整数	1	1.0.2.Final	持久性存活度检查的错误阈值
<code>registry.metrics.PersistenceExceptionLivenessCheck.statusResetWindowDurationSec</code>	整数	300	1.0.2.Final	持久性存活度检查的状态重置窗口持续时间

Name	类型	Default (默认)	可从以下位置获得	描述
registry.metrics.Persistenc eTimeoutReadinessCheck. counterResetWindowDurati onSec	整数	60	1.0.2.Final	持久性就绪度检查的计数 重置窗口持续时间
registry.metrics.Persistenc eTimeoutReadinessCheck. errorThreshold	整数	5	1.0.2.Final	持久性就绪度检查的错误 阈值
registry.metrics.Persistenc eTimeoutReadinessCheck. statusResetWindowDuratio nSec	整数	300	1.0.2.Final	持久性就绪度检查的状态 重置窗口持续时间
registry.metrics.Persistenc eTimeoutReadinessCheck.t imeoutSec	整数	15	1.0.2.Final	持久性就绪度检查超时
registry.metrics.Response ErrorLivenessCheck.count erResetWindowDurationSe c	整数	60	1.0.2.Final	响应存活度检查的计数重 置窗口持续时间
registry.metrics.Response ErrorLivenessCheck.disabl eLogging	布尔值	false	2.0.0.Final	禁用响应存活度检查的日 志记录
registry.metrics.Response ErrorLivenessCheck.errorT hreshold	整数	1	1.0.2.Final	响应存活度检查的错误阈 值
registry.metrics.Response ErrorLivenessCheck.status ResetWindowDurationSec	整数	300	1.0.2.Final	响应存活度检查的状态重 置窗口持续时间
registry.metrics.Response TimeoutReadinessCheck.c ounterResetWindowDurati onSec	instance<int eger>	60	1.0.2.Final	响应就绪度检查的计数重 置窗口持续时间
registry.metrics.Response TimeoutReadinessCheck.er rorThreshold	instance<int eger>	1	1.0.2.Final	响应就绪度检查的错误阈 值

Name	类型	Default (默认)	可从以下位置获得	描述
registry.metrics.ResponseTimeoutReadinessCheck.statusResetWindowDurationSec	instance<integer>	300	1.0.2.Final	响应就绪度检查的状态重置窗口持续时间
registry.metrics.ResponseTimeoutReadinessCheck.timeoutSec	instance<integer>	10	1.0.2.Final	响应就绪度检查的超时
registry.storage.metrics.cache.check-period	long	30000	2.1.0.Final	存储指标缓存检查周期

8.1.8. import

表 8.8. 导入配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
registry.import.url	可选<url>		2.1.0.Final	导入 URL

8.1.9. kafka

表 8.9. Kafka 配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
registry.events.kafka.topic	可选<string>		2.0.0.Final	事件 Kafka 主题
registry.events.kafka.topic-partition	可选<integer>		2.0.0.Final	事件 Kafka 主题分区

8.1.10. limits

表 8.10. 限制配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
registry.limits.config.max-artifact-labels	long	-1	2.2.3.Final	最大工件标签

Name	类型	Default (默认)	可从以下位置获得	描述
registry.limits.config.max-artifact-properties	long	-1	2.1.0.Final	最大工件属性
registry.limits.config.max-artifacts	long	-1	2.1.0.Final	最大工件数
registry.limits.config.max-description-length	long	-1	2.1.0.Final	最大工件描述长度
registry.limits.config.max-label-size	long	-1	2.1.0.Final	最大工件标签大小
registry.limits.config.max-name-length	long	-1	2.1.0.Final	最大工件名称长度
registry.limits.config.max-property-key-size	long	-1	2.1.0.Final	最大工件属性键大小
registry.limits.config.max-property-value-size	long	-1	2.1.0.Final	最大工件属性值大小
registry.limits.config.max-requests-per-second	long	-1	2.2.3.Final	每秒的最大工件请求
registry.limits.config.max-schema-size-bytes	long	-1	2.2.3.Final	最大模式大小 (字节)
registry.limits.config.max-total-schemas	long	-1	2.1.0.Final	最大总模式
registry.limits.config.max-versions-per-artifact	long	-1	2.1.0.Final	每个工件的最大版本
registry.storage.metrics.cache.max-size	long	1000	2.4.1.Final	存储指标缓存最大大小。

8.1.11. log

表 8.11. 日志配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
quarkus.log.level	string		2.0.0.Final	日志级别

8.1.12. redirects

表 8.12. 重定向配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
<code>registry.enable-redirects</code>	布尔值		2.1.2.Final	启用重定向
<code>registry.redirects</code>	<code>map<string, string></code>		2.1.2.Final	registry 重定向
<code>registry.url.override.host</code>	可选 <string>		2.5.0.Final	覆盖用于生成外部访问 URL 的主机名。当使用 HTTPS 透传 Ingress 或 Route 部署 Registry 时，主机和端口覆盖很有用。在这种情况下，用于重定向的请求 URL（和端口）不属于客户端使用的实际外部 URL，因为请求被代理。然后，重定向会失败，因为目标 URL 无法访问。
<code>registry.url.override.port</code>	可选 <integer>		2.5.0.Final	覆盖用于生成外部访问 URL 的端口。

8.1.13. rest

表 8.13. REST 配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
<code>registry.rest.artifact.deletion.enabled</code>	<code>boolean [dynamic]</code>	<code>false</code>	2.4.2-SNAPSHOT	启用工件版本删除
<code>registry.rest.artifact.download.maxSize</code>	<code>int</code>	<code>1000000</code>	2.2.6-SNAPSHOT	可以从 URL 下载的工件的最大大小
<code>registry.rest.artifact.download.skipSSLValidation</code>	布尔值	<code>false</code>	2.2.6-SNAPSHOT	从 URL 下载工件时跳过 SSL 验证

8.1.14. Store

表 8.14. 存储配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
artifacts.skip.disabled.latest	布尔值	true	2.4.2-SNAPSHOT	在检索最新的工件版本时跳过带有 DISABLED 状态的工件版本
quarkus.datasource.db-kind	string	postgresql	2.0.0.Final	DataSource Db kind
quarkus.datasource.jdbc.url	string		2.1.0.Final	DataSource jdbc URL
registry.sql.init	布尔值	true	2.0.0.Final	SQL init

8.1.15. ui

表 8.15. UI 配置选项

Name	类型	Default (默认)	可从以下位置获得	描述
quarkus.oidc.tenant-enabled	布尔值	false	2.0.0.Final	启用 UI OIDC 租户
registry.ui.config.apiUrl	string		1.3.0.Final	UI API URL
registry.ui.config.auth.oidc.client-id	string	none	2.2.6.Final	UI auth OIDC 客户端 ID
registry.ui.config.auth.oidc.redirect-url	string	none	2.2.6.Final	UI auth OIDC 重定向 URL
registry.ui.config.auth.oidc.url	string	none	2.2.6.Final	UI auth OIDC URL
registry.ui.config.auth.type	string	none	2.2.6.Final	UI 身份验证类型
registry.ui.config.uiCodegenEnabled	布尔值	true	2.4.2.Final	启用 UI codegen
registry.ui.config.uiContextPath	string	/ui/	2.1.0.Final	UI 上下文路径
registry.ui.features.readOnly	boolean [dynamic]	false	1.2.0.Final	UI 只读模式
registry.ui.features.settings	布尔值	false	2.2.2.Final	UI 功能设置

Name	类型	Default (默认)	可从以下位置 获得	描述
registry.ui.root	string		2.3.0.Final	覆盖 UI root 上下文（在 使用入站代理重新定位 UI 上下文时很有用）

附录 A. 使用您的订阅

Service Registry 通过软件订阅提供。要管理您的订阅，请访问红帽客户门户中的帐户。

访问您的帐户

1. 转至 access.redhat.com。
2. 如果您还没有帐户，请创建一个帐户。
3. 登录到您的帐户。

激活订阅

1. 转至 access.redhat.com。
2. 导航到 My Subscriptions。
3. 导航到 激活订阅 并输入您的 16 位激活号。

下载 ZIP 和 TAR 文件

要访问 ZIP 或 TAR 文件，请使用客户门户网站查找下载的相关文件。如果您使用 RPM 软件包，则不需要这一步。

1. 打开浏览器并登录红帽客户门户网站 产品下载页面，网址为 access.redhat.com/downloads。
2. 在 Integration 和 Automation 类别中找到 Red Hat Integration 条目。
3. 选择所需的 Service Registry 产品。此时会打开 Software Downloads 页面。
4. 单击组件的 Download 链接。

更新于 2024-01-26