OpenShift Container Platform 3.11

使用镜像

OpenShift Container Platform 3.11 使用镜像指南

Last Updated: 2023-03-25

OpenShift Container Platform 3.11 使用镜像

OpenShift Container Platform 3.11 使用镜像指南

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

使用这些主题找出 OpenShift Container Platform 3.11 用户可以使用哪些不同的 S2I(Source-to-Image)、数据库和 Docker 镜像。

目录

第 1 章 概述 6

第 2 章 SOURCE-TO-IMAGE(S2I) 7
2.1. 概述 7
2.2. JAVA 7
2.2.1. 概述 7
2.2.2. 版本 7
2.2.3. 镜像 7
2.2.4. 构建过程 7
2.2.5. Configuration 8
2.2.6. 构建和部署 Java 应用程序 8
2.2.7. 从源构建和部署 8
2.2.8. 从 Binary Artifacts 构建和部署 9
2.2.9. Java 环境变量 9
2.2.10. 其他资源 18
2.3. .NET CORE 18
2.3.1. 使用 .NET Core 的好处 18
2.3.2. 支持的版本 18
2.3.3. 镜像 18
2.3.4. 构建过程 18
2.3.5. 环境变量 19
2.3.6. 从.NET Core Source 快速部署应用程序 20
2.4. NODE.JS 21
2.4.1. 概述 21
2.4.2. 版本 21
2.4.3. 镜像 21
2.4.4. 构建过程 21
2.4.5. Configuration 22
2.4.6. 热部署 22
2.5. PERL 23
2.5.1. 概述 23
2.5.2. 版本 23
2.5.3. 镜像 23
2.5.4. 构建过程 24
2.5.5. Configuration 24
2.5.6. 访问日志 24
2.5.7. 热部署 25
2.6. PHP 25
2.6.1. 概述 25
2.6.2. 版本 25
2.6.3. 镜像 25
2.6.4. 构建过程 26
2.6.5. Configuration 26
2.6.5.1. Apache 配置 28
2.6.6. 访问日志 28
2.6.7. 热部署 28
2.7. PYTHON 28
2.7.1. 概述 28
2.7.2. 版本 28
2.7.3. 镜像 29
2.7.4. 构建过程 29

2.7.5. Configuration 29
2.7.6. 热部署 30
2.8. RUBY 31
2.8.1. 概述 31
2.8.2. 版本 31
2.8.3. 镜像 31
2.8.4. 构建过程 32
2.8.5. Configuration 32
2.8.6. 热部署 33
2.9. 自定义 S2I 镜像 34
2.9.1. 概述 34
2.9.2. 调用镜像中嵌入的脚本 34

第 3 章 数据库镜像 36
3.1. 概述 36
3.2. MYSQL 36
3.2.1. 概述 36
3.2.2. 版本 36
3.2.3. 镜像 36
3.2.4. 配置和使用 36
3.2.4.1. 初始化数据库 36
3.2.4.2. 在容器中运行 MySQL 命令 37
3.2.4.3. 环境变量 37
3.2.4.4. 卷挂载点 39
3.2.4.5. 更改密码 39
3.2.5. 从模板创建数据库服务 41
3.2.6. 使用 MySQL Replication 41
3.2.6.1. 为 MySQL Master 创建部署配置 41
3.2.6.2. 创建无标头服务 44
3.2.6.3. 扩展 MySQL 从服务器 45
3.2.7. 故障排除 45
3.2.7.1. Linux Native AIO 失败 45
3.3. POSTGRESQL 46
3.3.1. 概述 46
3.3.2. 版本 46
3.3.3. 镜像 46
3.3.4. 配置和使用 46
3.3.4.1. 初始化数据库 46
3.3.4.2. 在容器中运行 PostgreSQL 命令 47
3.3.4.3. 环境变量 47
3.3.4.4. 卷挂载点 48
3.3.4.5. 更改密码 48
3.3.5. 从模板创建数据库服务 49
3.4. MONGODB 50
3.4.1. 概述 50
3.4.2. 版本 50
3.4.3. 镜像 50
3.4.4. 配置和使用 51
3.4.4.1. 初始化数据库 51
3.4.4.2. 在容器中运行 MongoDB 命令 51
3.4.4.3. 环境变量 52
3.4.4.4. 卷挂载点 52
3.4.4.5. 更改密码 53

3.4.5. 从模板创建数据库服务 54
3.4.6. MongoDB 复制 54
3.4.6.1. 限制： 55
3.4.6.2. 使用示例模板 55
3.4.6.3. 扩展 56
3.4.6.4. 缩减 56
3.5. MARIADB 57
3.5.1. 概述 57
3.5.2. 版本 57
3.5.3. 镜像 57
3.5.4. 配置和使用 58
3.5.4.1. 初始化数据库 58
3.5.4.2. 在容器中运行 MariaDB 命令 58
3.5.4.3. 环境变量 58
3.5.4.4. 卷挂载点 60
3.5.4.5. 更改密码 61
3.5.5. 从模板创建数据库服务 62
3.5.6. 故障排除 62
3.5.6.1. Linux Native AIO 失败 62

第 4 章 其他镜像 64
4.1. 概述 64
4.2. JENKINS 64
4.2.1. 概述 64
4.2.2. 镜像 64
4.2.3. 配置和自定义 64
4.2.3.1. 身份验证 64
4.2.3.1.1. OpenShift Container Platform OAuth 身份验证 64
4.2.3.1.2. Jenkins 标准身份验证 65
4.2.3.2. 环境变量 65
4.2.3.3. 跨项目访问 67
4.2.3.4. 卷挂载点 68
4.2.3.5. 通过 Source-To-Image 自定义 Jenkins 镜像 68
4.2.3.6. 配置 Jenkins Kubernetes 插件 69
4.2.3.6.1. 权限注意事项 71
4.2.4. 使用方法 71
4.2.4.1. 从模板创建 Jenkins 服务 71
4.2.4.2. 使用 Jenkins Kubernetes 插件 72
4.2.4.3. 内存要求 74
4.2.5. Jenkins 插件 74
4.2.5.1. OpenShift Container Platform 客户端插件 74
4.2.5.2. OpenShift Container Platform Pipeline 插件 75
4.2.5.3. OpenShift Container Platform 同步插件 75
4.2.5.4. Kubernetes 插件 75
4.3. JENKINS 代理 75
4.3.1. 概述 75
4.3.2. 镜像 76
4.3.3. 配置和自定义 77
4.3.3.1. 环境变量 77
4.3.4. 使用方法 77
4.3.4.1. 内存要求 77
4.3.4.1.1. gradle 构建 78
4.3.5. 代理 Pod 保留 78

4.4. 其他容器镜像 79

第 1 章 概述

以下介绍了不同的 Source-to-Image (S2I)、数据库以及其他可供 OpenShift Container Platform 用户使用的容器镜像。

红帽官方容器镜像在 registry.redhat.io 上的 Red Hat Registry 中提供。OpenShift Container Platform 支持的 S2I、数据库和 Jenkins 镜像在 Red Hat Registry 的 **openshift3** 仓库中提供。例如，Atomic OpenShift Application Platform 镜像包括在 **registry.redhat.io/openshift3/ose**。

xPaaS 中间件镜像在 Red Hat Registry 上的相应产品存储库中提供，后缀为 **-openshift**。例如：JBoss EAP 镜像包括在 **registry.redhat.io/jboss-eap-6/eap64-openshift**。

本书涵盖的所有红帽支持的镜像均在 Red Hat Container Catalog 中介绍。如需了解每个镜像的各种版本，请查看其内容和使用方法详情。浏览或搜索您感兴趣的镜像。

重要

较新版容器镜像与较早版 OpenShift Container Platform 不兼容。根据您的 OpenShift Container Platform 版本，验证并使用正确的容器镜像版本。

第 2 章 SOURCE-TO-IMAGE(S2I)

2.1. 概述

本主题组包含不同 S2I(Source-to-Image) 支持的镜像的信息，可供 OpenShift Container Platform 用户使用。

2.2. JAVA

2.2.1. 概述

OpenShift Container Platform 为构建 Java 应用程序提供 S2I 构建器镜像。这些构建器镜像采用应用程序源或二进制工件，如果提供了源，请使用 Maven 构建源，并使用任何需要的依赖项组合工件，以创建一个包含 Java 应用程序的新、可直接运行的镜像。此生成的镜像可以在 OpenShift Container Platform 上运行，或者直接使用 Docker 运行。

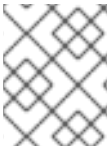
构建器镜像旨在与通过主类运行的基于 Maven- 的 Java 独立项目一起使用。

2.2.2. 版本

当前版本的 Java S2I 构建器镜像支持 OpenJDK 1.8 和 11、Jolokia 1.6.2 和 Maven 3.6。

2.2.3. 镜像

RHEL 7 和 RHEL 8 镜像可以通过 Red Hat Registry 提供。



注意

registry.redhat.io 需要身份验证。有关如何为 registry.redhat.io 配置您的环境的详情，请参考 Red Hat Container Registry 身份验证 。

基于 RHEL 7 的镜像

```
$ docker pull registry.redhat.io/redhat-openjdk-18/openjdk18-openshift
$ docker pull registry.redhat.io/openjdk/openjdk-11-rhel7
```

基于 RHEL 8 的镜像

```
$ docker pull registry.redhat.io/ubi8/openjdk-8
$ docker pull registry.redhat.io/ubi8/openjdk-11
```

要在 OpenShift Container Platform 上使用这些镜像，您可直接从 Red Hat Registry 访问 镜像，或将其推送(push)到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 镜像流定义。

2.2.4. 构建过程

S2I 通过将源代码注入容器，并为容器准备源代码来执行，从而生成可随时运行的镜像。它执行以下步骤：

1. 从构建器镜像启动容器。
2. 下载应用程序源代码。
3. 将脚本和应用程序源流传输到构建器镜像容器中。
4. 运行 *assemble* 脚本（从构建器镜像中）。
5. 保存最终镜像。

如需了解构建过程的详细概述，请参阅 S2I 构建过程。

2.2.5. Configuration

默认情况下，Java S2I 构建器镜像使用 Maven 来构建具有以下目标和选项的项目：

```
mvn -e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga -Dfabric8.skip=true --batch-mode
-Djava.net.preferIPv4Stack=true -s /tmp/artifacts/configuration/settings.xml -
Dmaven.repo.local=/tmp/artifacts/m2  package
```

基于这些默认值，构建器镜像将编译项目，并将所有传输依赖关系复制到输出目录中，而无需运行测试。另外，如果项目具有名为 **openshift** 的配置集，则会为构建激活它。

您可以通过指定以下环境变量来覆盖这些默认目标和选项：

变量名称	描述
MAVEN_S2I_ARTIFACT_DIRS	要扫描构建输出的源目录的相对路径，复制到 **DEPLOY_DIR**。默认为 **目标**。
JAVA_MAIN_CLASS	将 用作 **java** 的参数的主类。当给出这个环境变量时，**JAVA_APP_DIR** 中的所有 jar 文件都会添加到类路径和 **JAVA_LIB_DIR** 中。
MAVEN_ARGS	传递给 **mvn** 命令的参数。定义此变量会替换默认值，即 **-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga 软件包**。
MAVEN_ARGS_APPEND	额外的 Maven 参数。

这是配置 OpenJDK 容器行为的环境变量。有关完整的列表，请参阅 第 2.2.9 节 “Java 环境变量”。

2.2.6. 构建和部署 Java 应用程序

重要

必须首先 安装 OpenJDK 镜像流。如果您运行标准安装，则会包括镜像流。

同一 S2I 构建器镜像可用于从源或二进制工件构建 Java 应用程序。

2.2.7. 从源构建和部署

通过针对源存储库运行 **oc new-app**，可以使用 Java S2I 构建器镜像从源构建应用程序：

```
$ oc new-app registry.redhat.io/redhat-openjdk-18/openjdk18-openshift~https://github.com/jboss-openshift/openshift-quickstarts --context-dir=undertow-servlet
```

默认情况下不运行测试。要构建应用程序并运行测试作为构建的一部分，请覆盖默认的 **MAVEN_ARGS**，如下例所示：

```
$ oc new-app registry.redhat.io/redhat-openjdk-18/openjdk18-openshift~<git_repo_URL> --context-dir=<context_dir> --build-env='MAVEN_ARGS=-e -Popenshift -Dcom.redhat.xpaas.repo.redhatga package'
```

如果 Java 项目包含多个 Maven 模块，则显式指定工件输出目录会很有用。指定 Maven 项目输出工件的目录可让 S2I 构建提取它们。

要指定用于构建和工件输出目录的模块，请使用以下命令：

```
$ oc new-app registry.redhat.io/redhat-openjdk-18/openjdk18-openshift~<git_repo_URL> --context-dir=<context_dir> --build-env='MAVEN_S2I_ARTIFACT_DIRS=relative/path/to/artifacts/dir' --build-env='MAVEN_ARGS=install -pl <groupId>:<artifactId> -am'
```

2.2.8. 从 Binary Artifacts 构建和部署

您可以使用 Java S2I 构建器镜像来使用您提供的二进制工件来构建应用程序。

1. 创建一个新的二进制构建：

   ```
   $ oc new-build --name=<application_name> registry.redhat.io/redhat-openjdk-18/openjdk18-openshift --binary=true
   ```

2. 启动构建并指定本地机器上二进制工件的路径：

   ```
   $ oc start-build <application_name> --from-dir=/path/to/artifacts --follow
   ```

3. 创建应用程序：

   ```
   $ oc new-app <application_name>
   ```

2.2.9. Java 环境变量

下表提供了用于配置 OpenJDK 容器行为的 Java 环境变量的完整列表。

表 2.1. 配置环境变量

变量名称	描述	示例值

变量名称	描述	示例值
AB_JOLOKIA_CONFIG	如果设置，使用此文件（包括 path），作为 Jolokia JVM 代理属性，如 Jolokia 参考手册 中所述。如果没有设置，则使用 manual 中定义的设置创建 **/opt/jolokia/etc/jolokia.properties**。否则，本文档中的其他设置将被忽略。	**/opt/jolokia/custom.properties**
AB_JOLOKIA_DISCOVERY_ENABLED	启用 Jolokia 发现。默认值为 **false**。	**true**
AB_JOLOKIA_HOST	要绑定到的主机地址。默认为 **0.0.0.0**。	**127.0.0.1**
AB_JOLOKIA_ID	要使用的代理 ID。默认为 **$HOSTNAME**，即容器 ID。	**openjdk-app-1-xqlsj**
AB_JOLOKIA_OFF	如果设置，禁用 Jolokia 激活，例如，回显一个空值。默认情况下启用 Jolokia。	**true**
AB_JOLOKIA_OPTS	要附加到代理配置的额外选项。它们应该以 **key=value,key=value,...** 格式提供。	**backlog=20**
AB_JOLOKIA_PASSWORD	用于基本身份验证的密码。默认情况下，身份验证将被关闭。	**mypassword**
AB_JOLOKIA_PORT	要侦听的端口。默认为 **8778**。	**5432**
AB_JOLOKIA_USER	用于基本身份验证的用户。默认为 **jolokia**。	**myusername**
AB_PROMETHEUS_ENABLE	启用使用 Prometheus 代理。	**true**
AB_PROMETHEUS_JMX_EXPORTER_PORT	用于 Prometheus JMX Exporter 的端口。	**9799**
CONTAINER_CORE_LIMIT	CFS Bandwidth Control 中描述的计算内核限制。	**2**
CONTAINER_MAX_MEMORY	给定给容器的内存限值。	**1024**
GC_ADAPTIVE_SIZE_POLICY_WEIGHT	给定到当前的 GC 时间与之前的 GC 时间的权重。	**90**

变量名称	描述	示例值
GC_CONTAINER_OPTIONS	指定要使用的 Java GC。这个变量的值应该包含所需的 JRE 命令行界面选项，以指定所需的 GC，它会覆盖 **-XX:+UseParallelOldGC** 的默认值。	**-XX:+UseG1GC**
GC_MAX_HEAP_FREE_RATIO	GC 年后释放的最大堆百分比以避免缩小。	**40**
GC_MAX_METASPACE_SIZE	最大元空间大小。	**100**
GC_METASPACE_SIZE	初始的元空间大小。	**20**
GC_MIN_HEAP_FREE_RATIO	GC 年后释放的最小堆百分比以避免扩展。	**20**
GC_TIME_RATIO	指定垃圾回收外部花费的时间比率，例如应用程序执行的时间，以及垃圾回收中花费的时间。	**4**
HTTPS_PROXY	https 代理的位置。这优先于 **http_proxy** 和 **HTTP_PROXY**，用于 Maven 构建和 Java 运行时。	**myuser@127.0.0.1:8080**
HTTP_PROXY	http 代理的位置。这可用于 Maven 构建和 Java 运行时。	**127.0.0.1:8080**
JAVA_APP_DIR	应用程序所在的目录。应用程序中的所有路径都相对于这个目录。	**myapplication/`**
JAVA_ARGS	传递给 **java** 应用的参数。	-
JAVA_CLASSPATH	要使用的类路径。如果未提供，则启动脚本将检查文件 **JAVA_APP_DIR/classpath**，并以字面意义使用其内容作为类路径。如果这个文件不存在，则添加 app dir 中的所有 jars。(class**:JAVA_APP_DIR/***)。	-
JAVA_DEBUG	如果设置，则会切换远程调试。默认禁用此选项。	**true**
JAVA_DEBUG_PORT	用于远程调试的端口。默认为 **5005**。	**8787**

变量名称	描述	示例值
JAVA_DIAGNOSTICS	将这个变量设置为在发生数据时将一些诊断信息成为标准输出。默认禁用此选项。	**true**
JAVA_INITIAL_MEM_RATIO	在 **JAVA_OPTS** 中不提供 **-Xms** 选项。这用于根据最大堆大小计算默认初始堆内存。如果在容器没有内存约束的情况下在容器中使用，这个选项无效。如果有内存约束，则 **-Xms** 设置为此处设定的 **-Xmx** 内存的比率。默认值为 **25**，表示 **-Xmx** 的 25% 用作初始堆大小。您可以通过将此值设置为 **0** 来跳过此机制，在添加 no **-Xms** 选项时。	**25**
JAVA_LIB_DIR	保存 Java jar 文件和可选 **类路径** 的目录，该文件以单行路径（分开）或带有 jar 文件中列出的 jar 文件。如果没有设置，则 **JAVA_LIB_DIR** 与 **JAVA_APP_DIR** 相同。	-
JAVA_MAIN_CLASS	将 用作 **java** 的参数的主类。当给出这个环境变量时，**JAVA_APP_DIR** 中的所有 jar 文件都会添加到类路径和 **JAVA_LIB_DIR** 中。	**com.example.MainClass**
JAVA_MAX_INITIAL_MEM	在 **JAVA_OPTS** 中不提供 **-Xms** 选项。这用于计算初始堆内存的最大值。如果在容器没有内存约束的情况下在容器中使用，这个选项无效。如果有内存约束，则 **-Xms** 仅限于在此处设置的值。默认值为 **4096** MB，即 **-Xms** 的计算值不能超过 4096 MB。此变量的值以 MB 表示。	**4096**
JAVA_MAX_MEM_RATIO	在 **JAVA_OPTS** 中不提供 **-Xmx** 选项。这用于根据容器限制计算默认的 maximal 堆内存。如果在容器没有内存约束的情况下在容器中使用，这个选项无效。如果存在内存限制，**-Xmx** 会按照此处设定的容器可用内存比率。默认值为 **50**，这表示可用内存的 50% 用作上限。您可以通过将此值设置为 **0** 来跳过此机制，在添加了 **-Xmx** 选项时。	-

变量名称	描述	示例值
JAVA_OPTS	传递给 **java** 命令的 JVM 选项。	**-verbose:class**
JAVA_OPTS_APPEND	要附加到 **JAVA_OPTS** 中的选项的用户指定 Java 选项。	**-Dsome.property=foo**
LOGGING_SCRIPT_DEBUG	设置为 **true** 以启用脚本调试。弃用 **SCRIPT_DEBUG**。	**true**
MAVEN_ARGS	要在调用 Maven 时使用参数替换**默认软件包 hawt-app:build -DskipTests -e**.如果还没有绑定到 **软件包** 执行阶段，请确保运行 **hawt-app:build** 目标，否则启动脚本不起作用。	**-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package**
MAVEN_ARGS_APPEND	额外的 Maven 参数。	**-X -am -pl**
MAVEN_CLEAR_REPO	如果设置，则构建工件后会删除 Maven 存储库。这可用于使创建的应用程序镜像小，但会阻止增量构建。**S2I_ENABLE_INCREMENTAL_BUILDS** 将覆盖此变量。默认值为 **false**。	-
MAVEN_LOCAL_REPO	用作本地 Maven 存储库的目录。	**/home/jboss/.m2/repository**
MAVEN_MIRRORS	如果设置，则启用了多镜像支持，其他 **MAVEN_MIRROR_*** 变量带有前缀。例如，**DEV_ONE_MAVEN_MIRROR_URL** 和 **QE_TWO_MAVEN_MIRROR_URL**。	**dev-one,qe-two**
MAVEN_MIRROR_URL	用于检索工件的镜像的基本 URL。	**http://10.0.0.1:8080/repository/internal/**
MAVEN_REPOS	如果设置，则启用了多存储库支持，其他 **MAVEN_REPO_*** 变量带有前缀。例如，**DEV_ONE_MAVEN_REPO_URL** 和 **QE_TWO_MAVEN_REPO_URL**。	**dev-one,qe-two**

变量名称	描述	示例值
MAVEN_S2I_ARTIFACT_DIRS	要扫描构建输出的源目录的相对路径，复制到 **DEPLOY_DIR**。默认为 **目标**。	**目标**
MAVEN_S2I_GOALS	使用 maven 构建运行的空格分隔的目标列表。例如，**mvn $MAVEN_S2I_GOALS**。默认为 **软件包**。	**软件包安装**
MAVEN_SETTINGS_XML	要使用的自定义 Maven settings.xml 文件的位置。	**/home/jboss/.m2/settings.xml**
NO_PROXY	以逗号分隔的主机、IP 地址或可直接访问的域列表。这可用于 Maven 构建和 Java 运行时。	**foo.example.com,bar.example.com**
S2I_ARTIFACTS_DIR	用于工件的位置挂载与 **save-artifacts** 脚本保留，与增量构建一起使用。这不能被最终用户覆盖。	**${S2I_DESTINATION_DIR}/artifacts}**
S2I_DESTINATION_DIR	S2I 挂载的根目录，如 **io.openshift.s2i.destination** 标签指定。这不能被最终用户覆盖。	**/tmp**
S2I_ENABLE_INCREMENTAL_BUILDS	不要删除源和中间构建文件，以便可以保存它们以用于未来的构建。默认值为 **true**。	**true**
S2I_IMAGE_SOURCE_MOUNTS	源目录中的相对路径逗号分隔列表，应包含在镜像中。列表中可以包含通配符，这些通配符将使用 find 进行扩展。默认情况下，挂载目录的内容与源文件夹类似，其中 **S2I_SOURCE_CONFIGURATION_DIR** 的内容、**S2I_SOURCE_DATA_DIR** 以及 **S2I_SOURCE_DEPLOYMENTS_DIR** 复制到其对应的目标目录中。或者，如果 **install.sh** 文件位于挂载点的根目录中，它将执行。弃用 **CUSTOM_INSTALL_DIRECTORIES**。	**extras/***

变量名称	描述	示例值
S2I_SOURCE_CONFIGURATION_DIR	有关要复制到产品配置目录的目录的相对路径，请参阅 **S2I_TARGET_CONFIGURATION_DIR**。默认为 **配置**。	**配置**
S2I_SOURCE_DATA_DIR	有关要复制到产品数据目录的目录的相对路径，请参阅 **S2I_TARGET_DATA_DIR**。默认为 **数据**。	**data**
S2I_SOURCE_DEPLOYMENTS_DIR	包含要复制到产品部署目录的目录的相对路径，请参阅 **S2I_TARGET_DEPLOYMENTS_DIR**。默认为 **部署**。	**部署**
S2I_SOURCE_DIR	要构建源代码的位置。这不能被最终用户覆盖。	**${S2I_DESTINATION_DIR}/src}**
S2I_TARGET_CONFIGURATION_DIR	在 **S2I_SOURCE_DIR/S2I_SOURCE_CONFIGURATION_DIR** 中文件的绝对路径被复制。	**/opt/eap/standalone/configuration**
S2I_TARGET_DATA_DIR	复制 **S2I_SOURCE_DIR/S2I_SOURCE_DATA_DIR** 的绝对路径。	**/opt/eap/standalone/data**
S2I_TARGET_DEPLOYMENTS_DIR	在 **S2I_SOURCE_DIR/S2I_SOURCE_DEPLOYMENTS_DIR** 中文件的绝对路径。此外，这是复制构建输出的目录。	**/deployments**
http_proxy	http 代理的位置。这优先于 **HTTP_PROXY**，用于 Maven 构建和 Java 运行时。	**http://127.0.0.1:8080**
https_proxy	https 代理的位置。这优先于 **HTTPS_PROXY**、**http_proxy** 和 **HTTP_PROXY**，用于 Maven 构建和 Java 运行时。	**myuser:mypass@127.0.0.1:8080**
no_proxy	以逗号分隔的主机、IP 地址或可直接访问的域列表。这优先于 **NO_PROXY**，用于 Maven 构建和 Java 运行时。	***.example.com**

变量名称	描述	示例值
prefix_MAVEN_MIRROR_ID	用于指定镜像的 ID。如果省略，会生成一个唯一 ID。	**internal-mirror**
prefix_MAVEN_MIRROR_OF	通过此条目镜像的存储库 ID。默认为 **external：**	-
prefix_MAVEN_MIRROR_URL	镜像的 URL。	**http://10.0.0.1:8080/repository/internal**
prefix_MAVEN_REPO_DIRECTORY_PERMISSIONS	Maven 存储库目录权限。	**775**
prefix_MAVEN_REPO_FILE_PERMISSIONS	Maven 存储库文件权限。	**664**
prefix_MAVEN_REPO_HOST	如果不使用完全定义的 URL，则 Maven 存储库主机将回退到服务。	**repo.example.com**
prefix_MAVEN_REPO_ID	Maven 存储库 ID。	**my-repo-id**
prefix_MAVEN_REPO_LAYOUT	Maven 存储库布局.	**default**
prefix_MAVEN_REPO_NAME	Maven 存储库名称。	**my-repo-name**
prefix_MAVEN_REPO_PASSPHRASE	Maven 存储库密码短语。	**maven1!**
prefix_MAVEN_REPO_PASSWORD	Maven 存储库密码。	**maven1!**
prefix_MAVEN_REPO_PATH	Maven 存储库路径（如果没有使用完全定义的 URL）回退到服务。	**/maven2/**
prefix_MAVEN_REPO_PORT	Maven 存储库端口（如果没有使用完全定义的 URL）回退到服务。	**8080**
prefix_MAVEN_REPO_PRIVATE_KEY	Maven 存储库私钥。	**${user.home}/.ssh/id_dsa**
prefix_MAVEN_REPO_PROTOCOL	Maven 存储库协议（如果没有使用完全定义的 URL）回退到服务。	**http**
prefix_MAVEN_REPO_RELEASES_CHECKSUM_POLICY	Maven 存储库发布 checksum 策略。	**warn**

变量名称	描述	示例值
prefix_MAVEN_REPO_RELEASES_ENABLED	启用 Maven 存储库发行版本。	**true**
prefix_MAVEN_REPO_RELEASES_UPDATE_POLICY	Maven repository 发行更新策略。	**always**
prefix_MAVEN_REPO_SERVICE	如果没有指定 **prefix_MAVEN_REPO_URL**，用于查找 Maven 存储库服务。	**buscentr-myapp**
prefix_MAVEN_REPO_SNAPSHOTS_CHECKSUM_POLICY	Maven 存储库快照校验和策略。	**warn**
prefix_MAVEN_REPO_SNAPSHOTS_ENABLED	启用 Maven 存储库快照。	**true**
prefix_MAVEN_REPO_SNAPSHOTS_UPDATE_POLICY	Maven 存储库快照更新策略。	**always**
prefix_MAVEN_REPO_URL	Maven 存储库完全定义的 URL。	**http://repo.example.com:8080/maven2/**
prefix_MAVEN_REPO_USERNAME	Maven 存储库用户名。	**mavenUser**

表 2.2. 使用默认值配置环境变量

变量名称	描述	值
AB_JOLOKIA_AUTH_OPENSHIFT	切换到 OpenShift TLS 通信的客户端身份验证。此参数的值可以是相对可区分名称，它必须包含在出示客户端的证书中。启用此参数会自动将 Jolokia 切换到 https 通信模式。默认 CA 证书设置为 **/var/run/secrets/kubernetes.io/serviceaccount/ca.crt**。	**true**
AB_JOLOKIA_HTTPS	切换到安全与 https 的通信。默认情况下，如果在 **AB_JOLOKIA_OPTS** 中未提供 **serverCert** 配置，则会生成自签名证书。	**true**

变量名称	描述	值
AB_JOLOKIA_PASSWORD_RANDOM	确定是否生成随机 **AB_JOLOKIA_PASSWORD**。设置为 **true** 以生成随机密码。生成的值被写入 **/opt/jolokia/etc/jolokia.pw**。	**true**
AB_PROMETHEUS_JMX_EXPORTER_CONFIG	用于 Prometheus JMX Exporter 的配置路径。	**/opt/jboss/container/prometheus/etc/jmx-exporter-config.yaml**
S2I_SOURCE_DEPLOYMENTS_FILTER	复制部署时要应用空格分开的过滤器列表。默认为 *。	*

2.2.10. 其他资源

- 在红帽 JBoss 中间件 文档中查找更多信息和示例。

2.3. .NET CORE

2.3.1. 使用 .NET Core 的好处

.NET Core 是一个通用开发平台，它带有自动内存管理和现代编程语言。它允许用户有效地构建高质量的应用程序。.NET Core 可以通过认证的容器在 Red Hat Enterprise Linux（RHEL 7）和 OpenShift Container Platform 中找到。.NET Core 提供了：

- 采用基于微服务的方法，即某些组件使用 .NET 构建，其他组件则使用 Java 构建，但所有组件都可以在 Red Hat Enterprise Linux 和 OpenShift Container Platform 中常用支持的平台中运行。
- 在 Windows 中更轻松地开发新的 .NET Core 工作负载的容量；用户可以在 Red Hat Enterprise Linux 或者 Windows 服务器中部署并运行。
- 一个异构的数据中心，底层基础结构可以在不需要依赖 Windows 服务器的情况下运行 .NET 应用程序。
- 从 OpenShift Container Platform 内部访问许多流行开发框架，如 .NET、Java、Ruby 和 Python。

2.3.2. 支持的版本

.NET Core 生命周期 列出了当前支持的 .NET Core 版本。

2.3.3. 镜像

镜像可以通过 Red Hat Registry 提供。

如果您运行标准安装，则会包括 **dotnet** 镜像流。要包括最新支持的版本，您可以安装 .NET 镜像流。

2.3.4. 构建过程

S2I 通过将源代码注入容器，并为容器准备源代码来执行，从而生成可随时运行的镜像。它执行以下步骤：

1. 从构建器镜像启动容器。
2. 下载应用程序源代码。
3. 将脚本和应用程序源流传输到构建器镜像容器中。
4. 运行 *assemble* 脚本（从构建器镜像中）。
5. 保存最终镜像。

如需了解构建过程的详细概述，请参阅 S2I 构建过程。

2.3.5. 环境变量

.NET Core 镜像支持一些环境变量，您可以将其设置为控制 .NET Core 应用程序的构建行为。

注意

您必须设置控制 S2I 构建配置或 *.s2i/environment* 文件中构建行为的环境变量，使其可用于构建步骤。

表 2.3. NET Core 环境变量

变量名称	描述	默认
DOTNET_STARTUP_PROJECT	选择运行的项目。这必须是项目文件（例如： *csproj* 或 *fsproj* 或包含单个项目文件的文件夹。	.
DOTNET_ASSEMBLY_NAME	选择要运行的 assembly。 *不要*包括 **.dll** 扩展。把它设置为 *csproj*中指定的输出 assembly 名称（**PropertyGroup/AssemblyName**）。	*csproj* 文件的名称。
DOTNET_RESTORE_SOURCES	指定恢复操作中使用的 NuGet 软件包源，它是以空格分开的列表。这会覆盖 *NuGet.config* 文件中指定的所有源。	
DOTNET_TOOLS	指定在构建应用程序前要安装的 .NET 工具列表。要安装特定版本，请在软件包名称末尾添加 **@<version>**。	
DOTNET_NPM_TOOLS	指定在构建应用程序前要安装的 NPM 软件包列表。	
DOTNET_TEST_PROJECTS	指定要测试的测试项目列表。这必须是包含单个项目文件的项目文件或文件夹。对每个项目调用 **dotnet test**。	

变量名称	描述	默认
DOTNET_CONFIGURATION	以 **Debug** 或 **Release** 模式运行应用程序。这个值应该是 **Release** 或 **Debug**。	**Release**
DOTNET_VERBOSITY	指定 dotnet 构建命令的详细程度。设置后，环境变量会在构建开始时打印。这个变量可以被设置为 msbuild verbosity 值（**q[uiet]**、**m[inimal]**、**n[ormal]**、**d[etailed]** 和 **diag[nostic]**）。	
HTTP_PROXY, **HTTPS_PROXY**	配置构建和运行应用程序时使用的 HTTP/HTTPS 代理服务器。	
NPM_MIRROR	在构建过程中使用自定义 NPM registry 镜像下载软件包。	
ASPNETCORE_URLS	这个变量设定为 **http://*:8080** 以配置 ASP.NET Core 以使用由镜像公开的端口。*不建议*修改它。	**http://*:8080**
DOTNET_RM_SRC	当设置为 **true** 时，镜像中不包含源代码。	
DOTNET_SSL_DIRS	用于指定文件夹和文件列表，并附带要信任的额外 SSL 证书。证书由构建期间运行的每个进程以及构建后在镜像中运行的所有进程（包括构建的应用程序）信任。这些项可以是从 / 开始的绝对路径，也可以是源存储库中的路径（如证书）。	
DOTNET_RESTORE_DISABLE_PARALLEL	当设置为 **true** 时，会禁用并行恢复多个项目。这可减少构建容器以低 CPU 限值运行时恢复超时错误。	**false**
DOTNET_INCREMENTAL	当设置为 **true** 时，会保留 NuGet 软件包，以便将其用于增量构建。	**false**
DOTNET_PACK	当设置为 **true** 时，在 */opt/app-root/app.tar.gz* 中创建一个 *tar.gz* 文件，其中包含公布的应用程序。	

2.3.6. 从.NET Core Source 快速部署应用程序

重要

必须首先 安装 .NET 镜像流。如果您运行标准安装，则会包括镜像流。

镜像可用于通过针对示例存储库运行 **oc new-app** 来构建应用程序：

```
$ oc new-app dotnet:3.1~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-3.1 --
context-dir app
```

2.4. NODE.JS

2.4.1. 概述

OpenShift Container Platform 为构建和运行 Node.js 应用程序提供了 S2I 启用的 Node.js 镜像。Node.js S2I 构建器镜像将您的应用程序源与任何需要的依赖项组合在一起，以创建一个包含节点.js 应用程序的新镜像。此生成的镜像可由 OpenShift Container Platform 或容器运行时运行。

2.4.2. 版本

目前，OpenShift Container Platform 提供了 Node.js 的 0.10 、 4 和 6 版本。

2.4.3. 镜像

根据您的需要，这些镜像分为两种类型：

- RHEL 7
- CentOS 7

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/openshift3/nodejs-010-rhel7
$ docker pull registry.redhat.io/rhscl/nodejs-4-rhel7
```

基于 CentOS 7 的镜像

此镜像在 Docker Hub 上可用：

```
$ docker pull openshift/nodejs-010-centos7
```

要使用这些镜像，您可直接从镜像 registry 访问镜像，或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 ImageStream。您可以找到所有提供的 OpenShift Container Platform 镜像的示例镜像流定义。

2.4.4. 构建过程

S2I 通过将源代码注入容器，并为容器准备源代码来执行，从而生成可随时运行的镜像。它执行以下步骤：

1. 从构建器镜像启动容器。

2. 下载应用程序源代码。
3. 将脚本和应用程序源流传输到构建器镜像容器中。
4. 运行 *assemble* 脚本（从构建器镜像中）。
5. 保存最终镜像。

如需了解构建过程的详细概述，请参阅 S2I 构建过程。

2.4.5. Configuration

Node.js 镜像支持多个环境变量，这些变量可以被设置来控制 Node.js 运行时的配置和行为。

要将这些环境变量设置为镜像的一部分，您可以将它们放在源代码存储库内的一个 ***.s2i/environment*** 文件 中，或者在构建配置的 **sourceStrategy** 定义的 environment 部分 中定义它们。

您还可以设置在 创建新应用程序 时用于现有镜像的环境变量，或者 更新现有对象的环境变量，如部署配置。

注意

控制构建行为的环境变量必须作为 s2i 构建配置的一部分，或在 ***.s2i/environment*** 文件中设置，以便供构建步骤使用。

表 2.4. 开发模式环境变量

变量名称	描述
DEV_MODE	当设置为 **true** 时，启用热部署并打开 debug 端口。另外，指示使用工具指定镜像处于开发模式。默认为 **false**。
DEBUG_PORT	debug 端口。只有将 **DEV_MODE** 设为 true 时才有效。默认值为 5858。
NPM_MIRROR	自定义 NPM registry 镜像 URL。在构建过程中，所有 NPM 软件包都会从镜像链接下载。

2.4.6. 热部署

热部署允许您在不生成新的 S2I 构建的情况下迅速对应用程序进行和部署更改。要立即获取应用程序源代码中所做的更改，您必须使用 **DEV_MODE=true** 环境变量运行构建的镜像。

您可以在 创建新应用程序时设置新 环境变量，或 为现有对象更新环境变量。

警告

只在开发或调试时使用 **DEV_MODE=true** 环境变量。不建议在您的生产环境中使用此功能。

要更改正在运行的 pod 的源代码，请在容器中打开远程 shell:

```
$ oc rsh <pod_id>
```

进入正在运行的容器，更改您当前目录到 */opt/app-root/src*（源代码所在）。

2.5. PERL

2.5.1. 概述

OpenShift Container Platform 启用了 S2I 镜像来构建和运行 Perl 应用程序。Perl S2I 构建器镜像将您的应用程序源与任何需要的依赖项组合在一起，以创建一个包含 Perl 应用程序的新镜像。此生成的镜像可由 OpenShift Container Platform 或容器运行时运行。

2.5.2. 版本

目前，OpenShift Container Platform 支持 Perl 的版本 5.16、5.20 和 5.24。

2.5.3. 镜像

镜像根据您的需要分为两种类型：

- RHEL 7
- CentOS 7

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/openshift3/perl-516-rhel7
$ docker pull registry.redhat.io/rhscl/perl-520-rhel7
$ docker pull registry.redhat.io/rhscl/perl-524-rhel7
```

基于 CentOS 7 的镜像

Docker Hub 上提供了 Perl 5.16 的 CentOS 镜像：

```
$ docker pull openshift/perl-516-centos7
```

要使用这些镜像，您可直接从镜像 registry 访问镜像，或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 ImageStream。您可以找到所有提供的 OpenShift Container Platform 镜像的示例镜像流定义。

2.5.4. 构建过程

S2I 通过将源代码注入容器，并为容器准备源代码来执行，从而生成可随时运行的镜像。它执行以下步骤：

1. 从构建器镜像启动容器。
2. 下载应用程序源代码。
3. 将脚本和应用程序源流传输到构建器镜像容器中。
4. 运行 *assemble* 脚本（从构建器镜像中）。
5. 保存最终镜像。

如需了解构建过程的详细概述，请参阅 S2I 构建过程。

2.5.5. Configuration

Perl 镜像支持很多环境变量，可将其设置为控制 Perl 运行时的配置和行为。

要将这些环境变量设置为镜像的一部分，您可以将它们放在源代码存储库内的一个 ***.s2i/environment*** 文件 中，或者在构建配置的 **sourceStrategy** 定义的 environment 部分 中定义它们。

您还可以设置在 创建新应用程序 时用于现有镜像的环境变量，或者 更新现有对象的环境变量，如部署配置。

注意

控制构建行为的环境变量必须作为 s2i 构建配置的一部分，或在 ***.s2i/environment*** 文件中设置，以便供构建步骤使用。

表 2.5. Perl 环境变量

变量名称	描述
ENABLE_CPAN_TEST	当设置为 **true** 时，这个变量会安装所有的 cpan 模块并运行它们的测试。默认情况下，不测试模块。
CPAN_MIRROR	此变量指定 cpanminus 用来安装依赖项的镜像 URL。默认情况下不指定这个 URL。
PERL_APACHE2_RELOAD	把它设置为 **true** 来启用修改后的 Perl 模块的自动重新载入功能。默认情况下关闭自动重新载入。
HTTPD_START_SERVERS	StartServers 指令设定启动时创建的子服务器进程数目。默认值为 8。
HTTPD_MAX_REQUEST_WORKERS	Apache 可同时处理的请求数。默认值是 256，但如果内存有限，则会自动降低。

2.5.6. 访问日志

日志流数据会传输至标准输出，因此可使用 **oc logs** 命令查看它们。错误日志保存在 */tmp/error_log* 文件中，您可以使用 **oc rsh** 命令查看该文件来访问容器。

2.5.7. 热部署

热部署允许您在不生成新的 S2I 构建的情况下迅速对应用程序进行和部署更改。要在镜像中启用热部署，您必须将 **PERL_APACHE2_RELOAD** 环境变量设置为 **true**。例如，请参阅 **oc new-app** 命令。您可以使用 **oc set env** 命令更新现有对象的环境变量。

警告

您应该只在开发或调试时使用这个选项，我们不推荐在生产环境中启用这个选项。

要在运行的 pod 中更改源代码，请使用 **oc rsh** 命令进入容器：

```
$ oc rsh <pod_id>
```

进入正在运行的容器后，您的当前目录被设置为 */opt/app-root/src*（源代码所在的目录）。

2.6. PHP

2.6.1. 概述

OpenShift Container Platform 提供 已启用的 S2I 镜像用于构建和运行 PHP 应用程序。PHP S2I 构建器镜像将应用程序源与任何需要的依赖项组合在一起，以创建一个包含 PHP 应用程序的新镜像。此生成的镜像可由 OpenShift Container Platform 或容器运行时运行。

2.6.2. 版本

目前，OpenShift Container Platform 提供 PHP 版本 5.5、5.6 和 7.0 版本。

2.6.3. 镜像

根据您的需要，这些镜像分为两种类型：

- RHEL 7
- CentOS 7

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/openshift3/php-55-rhel7
$ docker pull registry.redhat.io/rhscl/php-56-rhel7
$ docker pull registry.redhat.io/rhscl/php-70-rhel7
```

基于 CentOS 7 的镜像

Docker Hub 上提供了 PHP 5.5 和 5.6 的 COS 镜像：

```
$ docker pull openshift/php-55-centos7
$ docker pull openshift/php-56-centos7
```

要使用这些镜像，您可直接从镜像 registry 访问镜像，或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用镜像流。

您可以找到所有提供的 OpenShift Container Platform 镜像的示例镜像流定义。

2.6.4. 构建过程

S2I 通过将源代码注入容器，并为容器准备源代码来执行，从而生成可随时运行的镜像。它执行以下步骤：

1. 从构建器镜像启动容器。
2. 下载应用程序源代码。
3. 将脚本和应用程序源流传输到构建器镜像容器中。
4. 运行 *assemble* 脚本（从构建器镜像中）。
5. 保存最终镜像。

如需了解构建过程的详细概述，请参阅 S2I 构建过程。

2.6.5. Configuration

PHP 镜像支持很多环境变量，可将其设置为控制 PHP 运行时的配置和行为。

要将这些环境变量设置为镜像的一部分，您可以将它们放在源代码存储库内的一个 ***.s2i/environment*** 文件 中，或者在构建配置的 **sourceStrategy** 定义的 environment 部分 中定义它们。

您还可以设置在 创建新应用程序 时用于现有镜像的环境变量，或者 更新现有对象的环境变量，如部署配置。

注意

控制构建行为的环境变量必须作为 s2i 构建配置的一部分，或在 ***.s2i/environment*** 文件中设置，以便供构建步骤使用。

以下环境变量在 ***php.ini*** 文件中设置与其对等的属性值：

表 2.6. PHP 环境变量

变量名称	描述	默认
ERROR_REPORTING	告知 PHP 错误、警告和您要为其操作的通知。	E_ALL & ~E_NOTICE

变量名称	描述	默认
DISPLAY_ERRORS	控制 PHP 输出错误、通知和警告的位置。	ON
DISPLAY_STARTUP_ERRORS	导致 PHP 启动序列中出现的任何显示错误与显示错误分开处理。	OFF
TRACK_ERRORS	将最后的错误/警告消息保存在 **$php_errormsg**（boolean）中。	OFF
HTML_ERRORS	将错误链接到与错误相关的文档。	ON
INCLUDE_PATH	PHP 源文件的路径。	*../opt/openshift/src:/opt/rh/php55/root/usr/share/pear*
SESSION_PATH	会话数据文件的位置。	*/tmp/sessions*
DOCUMENTROOT	为您的应用程序定义文档根目录的路径（例如：*/public*）。	*/*

以下环境变量在 *opcache.ini* 文件中设置其等同属性值：

表 2.7. 其他 PHP 设置

变量名称	描述	默认
OPCACHE_MEMORY_CONSUMPTION	OPcache 共享内存存储大小。	16M
OPCACHE_REVALIDATE_FREQ	检查更新的脚本时间戳的频率，以秒为单位。0 代表 OPcache 检查每个请求的更新。	2

您还可以通过设置来覆盖用于加载 PHP 配置的完整目录：

表 2.8. 其他 PHP 设置

变量名称	描述
PHPRC	设定到 *php.ini* 文件的路径。
PHP_INI_SCAN_DIR	扫描额外 *.ini* 配置文件的路径

您可以使用自定义的 composer 存储库镜像 URL 来下载软件包，而不是默认的 'packagist.org':

表 2.9. 组合器环境变量

变量名称	描述
COMPOSER_MIRROR	将这个变量设置为使用自定义编译程序存储库镜像 URL 在构建过程中下载所需的软件包。注：这只会影响 ***composer.json*** 中列出的软件包。

2.6.5.1. Apache 配置

如果应用程序的 **DocumentRoot** 位于源目录 */opt/openshift/src* 中，您可以提供自己的 *.htaccess* 文件来覆盖 Apache 默认行为，并指定如何处理应用程序请求。*.htaccess* 文件必须位于应用程序源的根目录下。

2.6.6. 访问日志

日志流数据会传输至标准输出，因此可使用 **oc logs** 命令查看它们。错误日志保存在 */tmp/error_log* 文件中，您可以使用 **oc rsh** 命令查看该文件来访问容器。

2.6.7. 热部署

热部署允许您在不生成新的 S2I 构建的情况下迅速对应用程序进行和部署更改。要立即获取应用程序源代码所做的更改，您必须使用 OPCACHE_REVALIDATE **_FREQ=0** 环境变量运行构建的镜像。

例如，请参阅 **oc new-app** 命令。您可以使用 **oc env** 命令更新现有对象的环境变量。

警告

您应该只在开发或调试时使用这个选项，我们不推荐在生产环境中启用这个选项。

要在运行的 pod 中更改源代码，请使用 **oc rsh** 命令进入容器：

```
$ oc rsh <pod_id>
```

进入正在运行的容器后，您的当前目录被设置为 */opt/app-root/src*（源代码所在的目录）。

2.7. PYTHON

2.7.1. 概述

OpenShift Container Platform 提供 S2I 启用 Python 镜像用于构建和运行 Python 应用程序。Python S2I 构建器镜像将您的应用程序源与任何需要的依赖项编译，以创建一个包含您的 Python 应用程序的新镜像。此生成的镜像可由 OpenShift Container Platform 或容器运行时运行。

2.7.2. 版本

目前，OpenShift Container Platform 提供 Python 版本 2.7、3.3、3.4 和 3.5。

2.7.3. 镜像

根据您的需要，这些镜像分为两种类型：

- RHEL 7
- CentOS 7

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/rhscl/python-27-rhel7
$ docker pull registry.redhat.io/openshift3/python-33-rhel7
$ docker pull registry.redhat.io/rhscl/python-34-rhel7
$ docker pull registry.redhat.io/rhscl/python-35-rhel7
```

基于 CentOS 7 的镜像

这些镜像在 Docker Hub 上可用：

```
$ docker pull centos/python-27-centos7
$ docker pull openshift/python-33-centos7
$ docker pull centos/python-34-centos7
$ docker pull centos/python-35-centos7
```

要使用这些镜像，您可直接从镜像 registry 访问镜像，或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 ImageStream。您可以找到所有提供的 OpenShift Container Platform 镜像的示例镜像流定义。

2.7.4. 构建过程

S2I 通过将源代码注入容器，并为容器准备源代码来执行，从而生成可随时运行的镜像。它执行以下步骤：

1. 从构建器镜像启动容器。
2. 下载应用程序源代码。
3. 将脚本和应用程序源流传输到构建器镜像容器中。
4. 运行 *assemble* 脚本（从构建器镜像中）。
5. 保存最终镜像。

如需了解构建过程的详细概述，请参阅 S2I 构建过程。

2.7.5. Configuration

Python 镜像支持很多可设定用来控制 Python 运行时的配置和行为的环境变量。

要将这些环境变量设置为镜像的一部分，您可以将它们放在源代码存储库内的一个 *.s2i/environment* 文件 中，或者在构建配置的 **sourceStrategy** 定义的 environment 部分 中定义它们。

您还可以设置在 创建新应用程序 时用于现有镜像的环境变量，或者 更新现有对象的环境变量，如部署配置。

注意

控制构建行为的环境变量必须作为 s2i 构建配置的一部分，或在 *.s2i/environment* 文件中设置，以便供构建步骤使用。

表 2.10. Python 环境变量

变量名称	描述
APP_FILE	这个变量指定传递给 Python 解释器的文件名，它负责启动应用程序。默认将此变量设置为 *app.py*。
APP_MODULE	这个变量指定了 WSGI 调用。它遵循模式 **$(MODULE_NAME):$(VARIABLE_NAME)**，其中模块名称是一个完整点路径，变量名称指的是指定模块中的某个功能。如果您使用 **setup.py** 安装应用程序，则可从该文件中读取模块名称，变量的默认值为 **application**。提供了 setup-test-app 示例。
APP_CONFIG	这个变量代表了到带有一个 gunicorn configuration 的有效 Python 文件的路径。
DISABLE_COLLECTSTATIC	把它设置为非空值，来限制在构建期间执行 **manage.py collectstatic**。只影响 Django 项目。
DISABLE_MIGRATE	将其设置为非空值，在限制在生成的镜像运行时执行 **manage.py migrate**。只影响 Django 项目。
PIP_INDEX_URL	将这个变量设置为使用自定义索引 URL 或镜像(mirror)在构建过程中下载所需的软件包。这只会影响 *requirements.txt* 文件中列出的软件包。
WEB_CONCURRENCY	设置它可更改 worker 数量的默认设置。默认情况下，它被设置为可用内核数的 4 倍。

2.7.6. 热部署

热部署允许您在不生成新的 S2I 构建的情况下迅速对应用程序进行和部署更改。如果您使用 Django，热部署可以正常工作。

要在使用 Gunicorn 的同时启用热部署，请确保在存储库中有一个 Gunicorn 配置文件，并将 **reload** 选项设置为 **true**。使用 **APP_CONFIG** 环境变量指定您的配置文件。例如，请参阅 **oc new-app** 命令。您可以使用 **oc set env** 命令更新现有对象的环境变量。

警告

您应该只在开发或调试时使用这个选项，我们不推荐在生产环境中启用这个选项。

要在运行的 pod 中更改源代码，请使用 **oc rsh** 命令进入容器：

```
$ oc rsh <pod_id>
```

进入正在运行的容器后，您的当前目录被设置为 */opt/app-root/src*（源代码所在的目录）。

2.8. RUBY

2.8.1. 概述

OpenShift Container Platform 提供 S2I 启用 Ruby 镜像来构建和运行 Ruby 应用程序。Ruby S2I 构建器镜像将应用程序源与任何需要的依赖项组合在一起，以创建一个包含 Ruby 应用程序的新镜像。此生成的镜像可由 OpenShift Container Platform 或容器运行时运行。

2.8.2. 版本

目前，OpenShift Container Platform 提供 Ruby 版本 2.0、2.2 和 2.3。

2.8.3. 镜像

根据您的需要，这些镜像分为两种类型：

- RHEL 7
- CentOS 7

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat registry 提供：

```
$ docker pull registry.redhat.io/openshift3/ruby-20-rhel7
$ docker pull registry.redhat.io/rhscl/ruby-22-rhel7
$ docker pull registry.redhat.io/rhscl/ruby-23-rhel7
```

基于 CentOS 7 的镜像

这些镜像在 Docker Hub 上可用：

```
$ docker pull openshift/ruby-20-centos7
$ docker pull openshift/ruby-22-centos7
$ docker pull centos/ruby-23-centos7
```

要使用这些镜像，您可直接从镜像 registry 访问镜像，或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 ImageStream。您可以找到所有提供

的 OpenShift Container Platform 镜像的示例镜像流定义。

2.8.4. 构建过程

S2I 通过将源代码注入容器，并为容器准备源代码来执行，从而生成可随时运行的镜像。它执行以下步骤：

1. 从构建器镜像启动容器。
2. 下载应用程序源代码。
3. 将脚本和应用程序源流传输到构建器镜像容器中。
4. 运行 *assemble* 脚本（从构建器镜像中）。
5. 保存最终镜像。

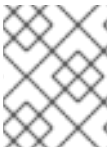
如需了解构建过程的详细概述，请参阅 S2I 构建过程。

2.8.5. Configuration

Ruby 镜像支持很多环境变量，它们可以被设置来控制 Ruby 运行时的配置和行为。

要将这些环境变量设置为镜像的一部分，您可以将它们放在源代码存储库内的一个 ***.s2i/environment*** 文件 中，或者在构建配置的 **sourceStrategy** 定义的 environment 部分 中定义它们。

您还可以设置在 创建新应用程序 时用于现有镜像的环境变量，或者 更新现有对象的环境变量，如部署配置。



注意

控制构建行为的环境变量必须作为 s2i 构建配置的一部分，或在 ***.s2i/environment*** 文件中设置，以便供构建步骤使用。

表 2.11. Ruby 环境变量

变量名称	描述
RACK_ENV	此变量指定部署 Ruby 应用程序的环境，例如：**production**、**development** 或 **test**根据日志详细程度、错误页和 ruby gem 安装，每个级别都有不同的行为。只有在将 **RACK_ENV** 设置为 **production** 时才会编译应用程序资产；默认值为 **production**。
RAILS_ENV	此变量指定 Ruby on Rails 应用程序的部署环境，例如：**production**、**development** 或 **test**。根据日志详细程度、错误页和 ruby gem 安装，每个级别都有不同的行为。只有在将 **RAILS_ENV** 设置为 **production** 时才会编译应用程序资产。默认将此变量设置为 **${RACK_ENV}**。
DISABLE_ASSET_COMPILATION	当设置为 **true** 时，这个变量会禁用资产编译过程。仅在应用程序在产品环境中运行时才会进行资产编译。因此，可以在资产已编译后使用此变量。

变量名称	描述
PUMA_MIN_THREADS, **PUMA_MAX_THREADS**	这个变量表示 Puma线程池中可用的最小和最大线程数。
PUMA_WORKERS	这个变量代表在 Puma 的 集群模式 中启动的 worker 进程数量（当 Puma 运行多个进程时）。如果没有明确设置，则默认行为将 **PUMA_WORKERS** 设置为适合容器可用内存和主机上内核数的值。
RUBYGEM_MIRROR	将这个变量设置为使用自定义 RubyGems 镜像 URL 在构建过程中下载所需的 gem 软件包。注：这个环境变量仅适用于 Ruby 2.2+ 镜像。

2.8.6. 热部署

热部署允许您在不生成新的 S2I 构建的情况下迅速对应用程序进行和部署更改。在这个镜像中启用热部署的方法因应用程序类型而异。

Ruby on Rails 应用程序

对于 Ruby on Rails 应用程序，使用传递给运行的 Pod 的 **RAILS_ENV=development** 环境变量运行构建 Rails 应用程序。对于现有的部署配置，可以使用 **oc set env** 命令：

```
$ oc set env dc/rails-app RAILS_ENV=development
```

其它类型 Ruby 应用程序（Sinatra、Padrino 等等）

对于其他类型的 Ruby 应用程序，应用程序必须使用 gem 构建，每次在运行的容器内更改源代码时都可以重新载入服务器。这些 gems 是：

- Shotgun
- Rerun
- Rack-livereload

为了能够在开发模式下运行应用程序，您必须修改 S2I ***run*** 脚本，以便网页服务器由所选 gem 启动，该 gem 会检查源代码中的更改。

使用 S2I ***run*** 脚本 的版本构建应用程序镜像后，使用 **RACK_ENV=development** 环境变量运行镜像。例如，请参阅 **oc new-app** 命令。您可以使用 **oc set env** 命令更新现有对象的环境变量。

警告

您应该只在开发或调试时使用这个选项，我们不推荐在生产环境中启用这个选项。

要在运行的 pod 中更改源代码，请使用 **oc rsh** 命令进入容器：

```
$ oc rsh <pod_id>
```

进入正在运行的容器后，您的当前目录被设置为 */opt/app-root/src*（源代码所在的目录）。

2.9. 自定义 S2I 镜像

2.9.1. 概述

S2I 构建器镜像通常包含 *assemble* 和 *run* 脚本，但这些脚本的默认行为可能并不适用于所有用户。本节介绍了一些自定义包含默认脚本的 S2I 构建器行为的方法。

2.9.2. 调用镜像中嵌入的脚本

通常，构建器镜像提供自己的 S2I 脚本版本，它适用于最常用的用例。如果这些脚本无法满足您的需要，S2I 提供了在 *.s2i/bin* 目录中添加自定义脚本覆盖它们的方法。但是，这样做代表完全替换了标准脚本。在某些情况下这是可以接受的，但在其他情况下，您可能更愿意在脚本之前（或之后）执行一些命令，同时保留镜像中提供的脚本逻辑。在这种情况下，您可以创建一个执行自定义逻辑的 wrapper 脚本，并将它进一步分配给镜像中的默认脚本。

要确定构建器镜像内的脚本位置，请查看 **io.openshift.s2i.scripts-url** 标签的值。使用 **docker inspect**:

```
$ docker inspect --format='{{ index .Config.Labels "io.openshift.s2i.scripts-url" }}' openshift/wildfly-100-centos7
image:///usr/libexec/s2i
```

您检查了 **openshift/wildfly-100-centos7** 构建器镜像，并发现这些脚本位于 */usr/libexec/s2i* 目录中。

使用这个知识，请从您自己的脚本中嵌套其调用。

例 2.1. *.s2i/bin/assemble* 脚本

```
#!/bin/bash
echo "Before assembling"

/usr/libexec/s2i/assemble
rc=$?

if [ $rc -eq 0 ]; then
    echo "After successful assembling"
else
    echo "After failed assembling"
fi

exit $rc
```

这个示例显示了一个自定义 *assemble* 脚本，它打印这个信息，从镜像中执行标准 *assemble* 脚本，并根据 *assemble* 脚本的退出代码打印另一个信息。

当嵌套 *run* 脚本时，您必须 使用 **exec** 来调用它来确保正确处理信号。以前，使用 **exec** 也无法在调用默认镜像运行脚本后运行附加命令。

例 2.2. *.s2i/bin/run* 脚本

```
#!/bin/bash
echo "Before running application"
exec /usr/libexec/s2i/run
```

第 3 章 数据库镜像

3.1. 概述

本节组包含可供 OpenShift Container Platform 用户使用的不同数据库镜像的信息。

注意

作为示例提供了为数据库镜像启用集群的配置，它们不适用于生产环境。

3.2. MYSQL

3.2.1. 概述

OpenShift Container Platform 为运行 Jenkins 提供容器镜像。该镜像可根据配置提供的用户名、密码和数据库名称设置提供数据库服务。

3.2.2. 版本

目前，OpenShift Container Platform 提供 MySQL 版本 5.6 和 5.7。

3.2.3. 镜像

该镜像根据您的需要分为两种类型：

- RHEL 7
- CentOS 7

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/rhscl/mysql-56-rhel7
$ docker pull registry.redhat.io/rhscl/mysql-57-rhel7
```

基于 CentOS 7 的镜像

MySQL 5.6 和 5.7 的 CentOS 镜像包括在 Docker Hub 中：

```
$ docker pull centos/mysql-56-centos7
$ docker pull centos/mysql-57-centos7
```

要使用这些镜像，您可直接从这些 registry 访问镜像或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 ImageStream。您可以找到所有提供的 OpenShift Container Platform 镜像的 ImageStream 定义 示例。

3.2.4. 配置和使用

3.2.4.1. 初始化数据库

您第一次使用共享卷时，数据库会与数据库管理员用户和 MySQL 根用户一起创建（如果您指定了 **MYSQL_ROOT_PASSWORD** 环境变量）。之后，MySQL 守护进程会启动。如果您要将卷重新关联到另一个容器，则不会创建数据库、数据库用户和管理员用户，并且会启动 MySQL 守护进程。

以下命令创建一个在容器中运行 MySQL 的新数据库 pod:

```
$ oc new-app \
    -e MYSQL_USER=<username> \
    -e MYSQL_PASSWORD=<password> \
    -e MYSQL_DATABASE=<database_name> \
    registry.redhat.io/rhscl/mysql-56-rhel7
```

3.2.4.2. 在容器中运行 MySQL 命令

OpenShift Container Platform 使用 Software Collections (SCLs)来安装和启动 MySQL。如果要在正在运行的容器内执行 MySQL 命令（用于调试），则必须使用 bash 调用它。

要做到这一点，首先确定 pod 的名称。例如，您可以查看当前项目中的 pod 列表：

```
$ oc get pods
```

然后，打开到 pod 的远程 shell 会话：

```
$ oc rsh <pod>
```

在进入容器时，会自动启用所需的 SCL。

您现在可以在 bash shell 中运行 **mysql** 命令来启动 MySQL 互动会话并执行普通的 MySQL 操作。例如，验证数据库用户：

```
bash-4.2$ mysql -u $MYSQL_USER -p$MYSQL_PASSWORD -h $HOSTNAME
$MYSQL_DATABASE
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.6.37 MySQL Community Server (GPL)
...
mysql>
```

完成后，输入 **quit** 或 **exit** 退出 MySQL 会话。

3.2.4.3. 环境变量

MySQL 用户名、密码和数据库名称必须使用以下环境变量进行配置：

表 3.1. MySQL 环境变量

变量名称	描述
MYSQL_USER	为您的应用程序创建的数据库用户指定用户名。
MYSQL_PASSWORD	**MYSQL_USER** 的密码。

变量名称	描述
MYSQL_DATABASE	**MYSQL_USER** 具有全部权利的数据库名称。
MYSQL_ROOT_PASSWORD	root 用户的密码（可选）。如果没有设置，则无法远程登录到 root 帐户。容器内的本地连接总是被允许，无需密码。
MYSQL_SERVICE_HOST	Kubernetes 自动创建的服务主机变量。
MYSQL_SERVICE_PORT	Kubernetes 自动创建的服务端口变量。

警告

您必须指定用户名、密码和数据库名称。如果没有指定全部三个变量，pod 将无法启动，OpenShift Container Platform 会不断尝试重启它。

MySQL 设置可使用以下环境变量进行配置：

表 3.2. 附加 MySQL 设置

变量名称	描述	默认
MYSQL_LOWER_CASE_TABLE_NAMES	设定表名称的存储和比较方式。	0
MYSQL_MAX_CONNECTIONS	允许客户端同时连接的最大数量。	151
MYSQL_MAX_ALLOWED_PACKET	一个数据包或生成/中间字符串的最大值。	200M
MYSQL_FT_MIN_WORD_LEN	FULLTEXT 索引中包含的单词的最小长度。	4
MYSQL_FT_MAX_WORD_LEN	FULLTEXT 索引中包含的单词的最大长度。	20
MYSQL_AIO	如果原生 AIO 出行问题，则控制 **innodb_use_native_aio** 设置值。	1
MYSQL_TABLE_OPEN_CACHE	所有线程打开的表的数量。	400

变量名称	描述	默认
MYSQL_KEY_BUFFER_SIZE	用于索引块的缓冲区的大小。	32M（或者 10% 可用内存）
MYSQL_SORT_BUFFER_SIZE	用于排序的缓冲区的大小。	256K
MYSQL_READ_BUFFER_SIZE	用于后续扫描的缓冲区的大小。	8M（或者 5% 可用内存）
MYSQL_INNODB_BUFFER_POOL_SIZE	InnoDB 缓存表和索引数据的缓冲池的大小。	32M（或者 50% 可用内存）
MYSQL_INNODB_LOG_FILE_SIZE	日志组中每个日志文件的大小。	8M（或者 15% 可用内存）
MYSQL_INNODB_LOG_BUFFER_SIZE	InnoDB 用来写入磁盘日志文件的缓冲大小。	8M（或者 15% 可用内存）

某些与内存相关的参数有两个默认值。当容器没有分配内存限值时使用这个固定值。其他值会根据可用内存在容器启动过程中动态计算。

3.2.4.4. 卷挂载点

可使用挂载卷运行 MySQL 镜像以便为数据库启用持久性存储：

- */var/lib/mysql/data* - 这是 MySQL 存储数据库文件的数据目录。

3.2.4.5. 更改密码

密码是镜像配置的一部分，因此唯一支持的为数据库用户（**MYSQL_USER**）和 root 用户更改密码的方法是分别更改环境变量 **MYSQL_PASSWORD** 和 **MYSQL_ROOT_PASSWORD**。

您可以通过在 web 控制台中查看 pod 或部署配置，或通过 CLI 列出环境变量来查看当前的密码：

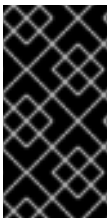
```
$ oc set env pod <pod_name> --list
```

当设置了 **MYSQL_ROOT_PASSWORD** 时，它就会为 **root** 用户启用远程访问，并可在有密码时启用远程访问。如果没有设置，会禁用 **root** 用户的远程访问。这不会影响通常具有远程访问权限的普通用户 **MYSQL_USER**。这也不会影响 root 用户的本地访问，**root** 用户总是可以在 **localhost** 中不带密码登录。

通过 SQL 语句或者通过环境变量更改数据库密码会导致变量中保存的值与实际密码不匹配。每当数据库容器启动时，它会将密码重置至环境变量中存储的值。

要更改这些密码，使用 **oc set env** 命令为相关的部署配置更新一个或多个所需的环境变量。如果多个部署配置使用这些环境变量，例如从模板创建应用程序时，您必须更新每个部署配置上的变量，以便密码可在所有方面同步。这可以在同一命令中完成：

```
$ oc set env dc <dc_name> [<dc_name_2> ...] \
  MYSQL_PASSWORD=<new_password> \
  MYSQL_ROOT_PASSWORD=<new_root_password>
```



重要

根据您的应用程序，应用程序的其它部分可能还会有其他密码的环境变量，它们也应该更新以匹配。例如，前端 pod 中可能会有一个更通用的 **DATABASE_USER** 变量，该变量应该与数据库用户的密码匹配。确保每个应用程序正在同步所有必需的环境变量，否则您的 pod 可能无法在触发时重新部署。

如果您 配置更改触发器，更新环境变量会触发数据库服务器的重新部署。否则，您必须手动启动新的部署以应用密码更改。

要验证新密码是否生效，首先请打开正在运行的 MySQL pod 的远程 shell 会话：

```
$ oc rsh <pod>
```

在 bash shell 中验证数据库用户的新密码：

```
bash-4.2$ mysql -u $MYSQL_USER -p<new_password> -h $HOSTNAME $MYSQL_DATABASE -te
"SELECT * FROM (SELECT database()) db CROSS JOIN (SELECT user()) u"
```

如果正确修改了密码，您应该可以看到类似如下的表：

```
+------------+--------------------+
| database() | user()             |
+------------+--------------------+
| sampledb   | user0PG@172.17.42.1 |
+------------+--------------------+
```

验证 **root** 用户的新密码：

```
bash-4.2$ mysql -u root -p<new_root_password> -h $HOSTNAME $MYSQL_DATABASE -te
"SELECT * FROM (SELECT database()) db CROSS JOIN (SELECT user()) u"
```

如果正确修改了密码，您应该可以看到类似如下的表：

```
+------------+-----------------+
| database() | user()          |
+------------+-----------------+
| sampledb   | root@172.17.42.1 |
+------------+-----------------+
```

3.2.5. 从模板创建数据库服务

OpenShift Container Platform 提供了一个 模板，可简化新数据库服务的创建。模板提供参数字段来定义所有强制环境变量（用户、密码、数据库名称等）,并使用预先定义的默认值（包括密码值的自动生成）。它还将定义 部署配置 和 服务。

在初始集群设置过程中，您的集群管理员应该在默认 **openshift** 项目中注册 MySQL 模板。如需了解更多详细信息，请参阅载入默认镜像流和模板。

有两个可用模板：

- **MySQL-ephemeral** 只是用于开发或测试目的，因为它对数据库内容使用临时存储。这意味着，如果数据库 pod 因某种原因被重启，如 pod 正在移至另一节点，或正在更新和触发重新部署的部署配置，则所有数据将会丢失。
- **MySQL-persistent** 将持久性卷存储用于数据库数据，这意味着 pod 重启后数据会保留下来。使用持久性卷需要在 OpenShift Container Platform 部署中定义持久性卷池。设置池的信息包括在使用 NFS 的持久性存储 中。

您可以按照以下步骤实例化模板 https://access.redhat.com/documentation/en-us/openshift_container_platform/3.11/html-single/developer_guide/#dev-guide-templates。

在实例化该服务后，您可以将用户名、密码和数据库名称环境变量复制到旨在访问数据库的另一个组件的部署配置中。然后该组件可以通过定义的服务访问数据库。

3.2.6. 使用 MySQL Replication

注意

作为示例提供了为数据库镜像启用集群的配置，它们不适用于生产环境。

红帽提供了一个 MySQL master-slave 复制（集群）概念验证 模板; 您可以 从 GitHub 获取示例模板。

将示例模板上传到当前项目的模板库中：

```
$ oc create -f \
    https://raw.githubusercontent.com/sclorg/mysql-
container/master/examples/replica/mysql_replica.json
```

以下小节详细介绍了示例模板中定义的对象，并描述了它们如何一起启动实施 master-slave 复制的 MySQL 服务器集群。这是推荐的 MySQL 复制策略。

3.2.6.1. 为 MySQL Master 创建部署配置

要设置 MySQL 复制，在定义 复制控制器 的示例模板中定义一个部署配置。对于 MySQL master-slave 复制，需要两个部署配置。一个部署配置定义 MySQL *master* 服务器，再定义 MySQL 的 *从* 服务器。

要让 MySQL 服务器充当 master，必须将部署配置中容器定义中的 **command** 字段设置为 **run-mysqld-master**。这个脚本作为 MySQL 镜像的替代入口点，并将 MySQL 服务器配置为复制时作为主服务器运行。

MySQL 复制需要一个特殊的用户来转发主服务器和辅服务器之间的数据。模板中为此定义了以下环境变量：

变量名称	描述	默认
MYSQL_MASTER_USER	复制用户的用户名	**master**
MYSQL_MASTER_PASSWORD	复制用户的密码	**generated**

例 3.1. 示例模板中的 MySQL 主部署配置对象定义

```
kind: "DeploymentConfig"
apiVersion: "v1"
metadata:
  name: "mysql-master"
spec:
  strategy:
    type: "Recreate"
  triggers:
    - type: "ConfigChange"
  replicas: 1
  selector:
    name: "mysql-master"
  template:
    metadata:
      labels:
        name: "mysql-master"
    spec:
      volumes:
        - name: "mysql-master-data"
          persistentVolumeClaim:
            claimName: "mysql-master"
      containers:
        - name: "server"
          image: "openshift/mysql-56-centos7"
          command:
            - "run-mysqld-master"
          ports:
            - containerPort: 3306
              protocol: "TCP"
          env:
            - name: "MYSQL_MASTER_USER"
              value: "${MYSQL_MASTER_USER}"
            - name: "MYSQL_MASTER_PASSWORD"
              value: "${MYSQL_MASTER_PASSWORD}"
            - name: "MYSQL_USER"
```

```
          value: "${MYSQL_USER}"
        - name: "MYSQL_PASSWORD"
          value: "${MYSQL_PASSWORD}"
        - name: "MYSQL_DATABASE"
          value: "${MYSQL_DATABASE}"
        - name: "MYSQL_ROOT_PASSWORD"
          value: "${MYSQL_ROOT_PASSWORD}"
      volumeMounts:
        - name: "mysql-master-data"
          mountPath: "/var/lib/mysql/data"
      resources: {}
      terminationMessagePath: "/dev/termination-log"
      imagePullPolicy: "IfNotPresent"
      securityContext:
        capabilities: {}
        privileged: false
  restartPolicy: "Always"
  dnsPolicy: "ClusterFirst"
```

由于我们声明在这个部署配置中有一个持久性卷来为 MySQL master 服务器保留所有数据，所以必须请集群管理员创建一个可以声明存储的持久性卷。

创建部署配置以及启动 MySQL master 服务器的 pod 后，它会创建 **MYSQL_DATABASE** 定义的数据库，并将服务器配置为将这个数据库复制到辅服务器。

这个示例只定义 MySQL 主服务器的一个副本。这会导致 OpenShift Container Platform 只启动一个服务器实例。不支持多个实例（multi-master），因此您无法扩展此复制控制器。

要复制由 MySQL master 创建的数据库，模板中定义了一个部署配置。这个部署配置会创建一个复制控制器，它会使用将 **command** 字段设置为 **run-mysqld-slave** 来启动 MySQL 镜像。这个替代的入口点跳过数据库初始化并将 MySQL 服务器配置为连接到 **mysql-master** 服务，在示例模板中也定义了这个服务。

例 3.2. 示例模板中的 MySQL 辅助部署配置对象定义

```
kind: "DeploymentConfig"
apiVersion: "v1"
metadata:
  name: "mysql-slave"
spec:
  strategy:
    type: "Recreate"
  triggers:
    - type: "ConfigChange"
  replicas: 1
  selector:
    name: "mysql-slave"
  template:
    metadata:
      labels:
        name: "mysql-slave"
    spec:
      containers:
        - name: "server"
          image: "openshift/mysql-56-centos7"
```

```
        command:
          - "run-mysqld-slave"
        ports:
          - containerPort: 3306
            protocol: "TCP"
        env:
          - name: "MYSQL_MASTER_USER"
            value: "${MYSQL_MASTER_USER}"
          - name: "MYSQL_MASTER_PASSWORD"
            value: "${MYSQL_MASTER_PASSWORD}"
          - name: "MYSQL_DATABASE"
            value: "${MYSQL_DATABASE}"
        resources: {}
        terminationMessagePath: "/dev/termination-log"
        imagePullPolicy: "IfNotPresent"
        securityContext:
          capabilities: {}
          privileged: false
      restartPolicy: "Always"
      dnsPolicy: "ClusterFirst"
```

这个示例部署配置启动复制控制器，并将初始副本数设置为 1。您可以双向扩展此复制控制器，受帐户的资源容量的限制。

3.2.6.2. 创建无标头服务

MySQL 辅助复制控制器创建的 pod 必须访问 MySQL 主服务器才能注册以复制。示例模板定义了名为 **mysql-master** 的无标头服务。这个服务不仅仅用于复制，客户端也可以将查询发送到 MySQL 主机的 **mysql-master:3306**。

要具有无头服务，服务定义中的 **clusterIP** 参数设置为 **None**。然后，您可以使用 DNS 查询获取代表该服务的当前端点的 Pod IP 地址列表。

例 3.3. 示例模板中的无标头服务对象定义

```
kind: "Service"
apiVersion: "v1"
metadata:
  name: "mysql-master"
  labels:
    name: "mysql-master"
spec:
  ports:
    - protocol: "TCP"
      port: 3306
      targetPort: 3306
      nodePort: 0
  selector:
    name: "mysql-master"
  clusterIP: "None"
  type: "ClusterIP"
  sessionAffinity: "None"
status:
  loadBalancer: {}
```

3.2.6.3. 扩展 MySQL 从服务器

增加集群中的成员数量:

```
$ oc scale rc mysql-slave-1 --replicas=<number>
```

这样可让 复制控制器 创建新的 MySQL 从 Pod。当创建新的从系统时，从系统入口点首先会尝试联系 **mysql-master** 服务，并将自己注册到复制集合中。完成后，MySQL 主服务器会发送复制数据库的辅号。

缩减时，MySQL 从系统将会关闭，并且因为从系统没有定义任何持久性存储，所以从系统中的所有数据都会丢失。然后 MySQL 主服务器会发现从系统无法再访问，它会自动从复制中移除它。

3.2.7. 故障排除

本节描述了您可能会遇到的问题，并给出可能的解决方案。

3.2.7.1. Linux Native AIO 失败

症状

MySQL 容器无法启动，日志显示类似如下：

```
151113  5:06:56 InnoDB: Using Linux native AIO
151113  5:06:56  InnoDB: Warning: io_setup() failed with EAGAIN. Will make 5 attempts before
giving up.
InnoDB: Warning: io_setup() attempt 1 failed.
InnoDB: Warning: io_setup() attempt 2 failed.
Waiting for MySQL to start ...
InnoDB: Warning: io_setup() attempt 3 failed.
InnoDB: Warning: io_setup() attempt 4 failed.
Waiting for MySQL to start ...
InnoDB: Warning: io_setup() attempt 5 failed.
151113  5:06:59  InnoDB: Error: io_setup() failed with EAGAIN after 5 attempts.
InnoDB: You can disable Linux Native AIO by setting innodb_use_native_aio = 0 in my.cnf
151113  5:06:59 InnoDB: Fatal error: cannot initialize AIO sub-system
151113  5:06:59 [ERROR] Plugin 'InnoDB' init function returned error.
151113  5:06:59 [ERROR] Plugin 'InnoDB' registration as a STORAGE ENGINE failed.
151113  5:06:59 [ERROR] Unknown/unsupported storage engine: InnoDB
151113  5:06:59 [ERROR] Aborting
```

解释

MySQL 的存储引擎因为资源限制而无法使用内核的 AIO（异步 I/O）功能。

解决方案

通过将环境变量 **MYSQL_AIO** 设置为 **0** 来完全关闭 AIO 的使用。在后续的部署中，这会让 MySQL 配置变量 **innodb_use_native_aio** 的值为 **0**。

另外，还可增加 **aio-max-nr** 内核资源。下面的例子检查了 **aio-max-nr** 的当前值并加倍。

```
$ sysctl fs.aio-max-nr
fs.aio-max-nr = 1048576
# sysctl -w fs.aio-max-nr=2097152
```

这会针对每个节点进行解析，持续到下一个节点重启为止。

3.3. POSTGRESQL

3.3.1. 概述

OpenShift Container Platform 为运行 PostgreSQL 提供容器镜像。该镜像可根据配置提供的用户名、密码和数据库名称设置提供数据库服务。

3.3.2. 版本

目前，OpenShift Container Platform 支持 PostgreSQL 的版本是 9.4 和 9.5。

3.3.3. 镜像

根据您的需要，这些镜像分为两种类型：

- RHEL 7
- CentOS 7

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/rhscl/postgresql-94-rhel7
$ docker pull registry.redhat.io/rhscl/postgresql-95-rhel7
```

基于 CentOS 7 的镜像

这些镜像在 Docker Hub 上可用：

```
$ docker pull centos/postgresql-94-centos7
$ docker pull centos/postgresql-95-centos7
```

要使用这些镜像，您可直接从这些 registry 访问镜像或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 ImageStream。您可以找到所有提供的 OpenShift Container Platform 镜像的 ImageStream 定义 示例。

3.3.4. 配置和使用

3.3.4.1. 初始化数据库

您第一次使用共享卷时，数据库会与数据库管理员用户和 PostgreSQL postgres 用户一起创建（如果您指定了 **POSTGRESQL_ADMIN_PASSWORD** 环境变量）。之后，PostgreSQL 守护进程会启动。如果您要将卷重新关联到另一个容器，则不会创建数据库、数据库用户和管理员用户，并且会启动 PostgreSQL 守护进程。

以下命令创建使用容器中运行 PostgreSQL 的新数据库 pod:

```
$ oc new-app \
    -e POSTGRESQL_USER=<username> \
    -e POSTGRESQL_PASSWORD=<password> \
    -e POSTGRESQL_DATABASE=<database_name> \
    registry.redhat.io/rhscl/postgresql-95-rhel7
```

3.3.4.2. 在容器中运行 PostgreSQL 命令

OpenShift Container Platform 使用 Software Collections (SCLs)来安装和启动 PostgreSQL。如果要在正在运行的容器内执行 PostgreSQL 命令（用于调试），则必须使用 bash 调用它。

要做到这一点，首先确定正在运行的 PostgreSQL Pod 的名称。例如，您可以查看当前项目中的 pod 列表：

```
$ oc get pods
```

然后，打开到所需 pod 的远程 shell 会话：

```
$ oc rsh <pod>
```

在进入容器时，会自动启用所需的 SCL。

现在，可以在 bash shell 中运行 **psql** 命令来启动 PostgreSQL 互动会话并执行常规 PostgreSQL 操作。例如，验证数据库用户：

```
bash-4.2$ PGPASSWORD=$POSTGRESQL_PASSWORD psql -h postgresql
$POSTGRESQL_DATABASE $POSTGRESQL_USER
psql (9.5.16)
Type "help" for help.

default=>
```

完成后，输入 **\q** 退出 PostgreSQL 会话。

3.3.4.3. 环境变量

PostgreSQL 用户名、密码和数据库名称必须使用以下环境变量进行配置：

表 3.3. PostgreSQL 环境变量

变量名称	描述
POSTGRESQL_USER	要创建的 PostgreSQL 帐户的用户名。这个用户对数据库有完全的权利。
POSTGRESQL_PASSWORD	用户帐户的密码。
POSTGRESQL_DATABASE	数据库名称。

变量名称	描述
POSTGRESQL_ADMIN_PASSWORD	postgres 管理员用户的可选密码。如果没有设置，则无法远程登录 **postgres** 帐户。容器内的本地连接总是被允许，无需密码。

警告

您必须指定用户名、密码和数据库名称。如果没有指定全部三个变量，pod 将无法启动，OpenShift Container Platform 会不断尝试重启它。

PostgreSQL 设置可使用以下环境变量进行配置：

表 3.4. 其他 PostgreSQL 设置

变量名称	描述	默认
POSTGRESQL_MAX_CONNECTIONS	允许的最大客户端连接数。	100
POSTGRESQL_MAX_PREPARED_TRANSACTIONS	处于 "prepared" 状态的事务的最大数量。如果使用准备的事务，其值应至少与 **POSTGRESQL_MAX_CONNECTIONS** 相同。	0
POSTGRESQL_SHARED_BUFFERS	专用于 PostgreSQL 缓存数据的内存量。	32M
POSTGRESQL_EFFECTIVE_CACHE_SIZE	预计由操作系统和 PostgreSQL 本身用于磁盘缓存的内存量。	128M

3.3.4.4. 卷挂载点

PostgreSQL 镜像可以使用挂载卷运行，以便为数据库启用持久性存储：

- */var/lib/pgsql/data* - 这是 PostgreSQL 存储数据库文件的数据库集群目录。

3.3.4.5. 更改密码

密码是镜像配置的一部分，因此唯一支持的为数据库用户(**POSTGRESQL_USER**)和 **postgres** 管理员用户更改密码的方法是分别更改环境变量 **POSTGRESQL_PASSWORD** 和 **POSTGRESQL_ADMIN_PASSWORD**。

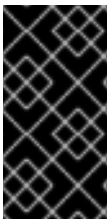
您可以通过在 web 控制台中查看 pod 或部署配置，或通过 CLI 列出环境变量来查看当前的密码：

```
$ oc set env pod <pod_name> --list
```

通过 SQL 语句或者通过环境变量更改数据库密码会导致变量中保存的值与实际密码不匹配。每当数据库容器启动时，它会将密码重置至环境变量中存储的值。

要更改这些密码，使用 **oc set env** 命令为相关的部署配置更新一个或多个所需的环境变量。如果多个部署配置使用这些环境变量，例如从模板创建应用程序时，您必须更新每个部署配置上的变量，以便密码可在所有方面同步。这可以在同一命令中完成：

```
$ oc set env dc <dc_name> [<dc_name_2> ...] \
  POSTGRESQL_PASSWORD=<new_password> \
  POSTGRESQL_ADMIN_PASSWORD=<new_admin_password>
```



重要

根据您的应用程序，应用程序的其它部分可能还会有其他密码的环境变量，它们也应该更新以匹配。例如，前端 pod 中可能会有一个更通用的 **DATABASE_USER** 变量，该变量应该与数据库用户的密码匹配。确保每个应用程序正在同步所有必需的环境变量，否则您的 pod 可能无法在触发时重新部署。

如果您 配置更改触发器，更新环境变量会触发数据库服务器的重新部署。否则，您必须手动启动新的部署以应用密码更改。

要验证新密码是否有效，首先请打开一个远程 shell 会话到正在运行的 PostgreSQL pod:

```
$ oc rsh <pod>
```

在 bash shell 中验证数据库用户的新密码：

```
bash-4.2$ PGPASSWORD=<new_password> psql -h postgresql $POSTGRESQL_DATABASE
$POSTGRESQL_USER -c "SELECT * FROM (SELECT current_database()) cdb CROSS JOIN
(SELECT current_user) cu"
```

如果正确修改了密码，您应该可以看到类似如下的表：

```
 current_database | current_user
------------------+--------------
 default          | django
(1 row)
```

在 bash shell 中验证 **postgres** 管理员用户的新密码：

```
bash-4.2$ PGPASSWORD=<new_admin_password> psql -h postgresql
$POSTGRESQL_DATABASE postgres -c "SELECT * FROM (SELECT current_database()) cdb
CROSS JOIN (SELECT current_user) cu"
```

如果正确修改了密码，您应该可以看到类似如下的表：

```
 current_database | current_user
------------------+--------------
 default          | postgres
(1 row)
```

3.3.5. 从模板创建数据库服务

OpenShift Container Platform 提供了一个 模板，可简化新数据库服务的创建。模板提供参数字段来定义所有强制环境变量（用户、密码、数据库名称等）,并使用预先定义的默认值（包括密码值的自动生成）。它还将定义 部署配置 和 服务。

在初始集群设置过程中，您的集群管理员应该在默认 **openshift** 项目中注册 PostgreSQL 模板。如需了解更多详细信息，请参阅载入默认镜像流和模板。

有两个可用模板：

- **PostgreSQL-ephemeral** 仅用于开发或测试目的，因为它对数据库内容使用临时存储。这意味着，如果数据库 pod 因某种原因被重启，如 pod 正在移至另一节点，或正在更新和触发重新部署的部署配置，则所有数据将会丢失。
- **PostgreSQL-persistent** 将持久性卷存储用于数据库数据，这意味着 pod 重启后数据会保留下来。使用持久性卷需要在 OpenShift Container Platform 部署中定义持久性卷池。设置池的信息包括在使用 NFS 的持久性存储 中。

您可以按照以下步骤实例化模板 https://access.redhat.com/documentation/en-us/openshift_container_platform/3.11/html-single/developer_guide/#dev-guide-templates。

在实例化该服务后，您可以将用户名、密码和数据库名称环境变量复制到旨在访问数据库的另一个组件的部署配置中。然后该组件可以通过定义的服务访问数据库。

3.4. MONGODB

3.4.1. 概述

OpenShift Container Platform 为运行 MongoDB 提供容器镜像。该镜像可根据配置提供的用户名、密码和数据库名称设置提供数据库服务。

3.4.2. 版本

目前，OpenShift Container Platform 提供了 MongoDB 的 2.6、3.2 和 3.4 版本。

3.4.3. 镜像

根据您的需要，这些镜像分为两种类型：

- RHEL 7
- CentOS 7

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/rhscl/mongodb-26-rhel7
$ docker pull registry.redhat.io/rhscl/mongodb-32-rhel7
$ docker pull registry.redhat.io/rhscl/mongodb-34-rhel7
```

基于 CentOS 7 的镜像

这些镜像在 Docker Hub 上可用：

```
$ docker pull centos/mongodb-26-centos7
$ docker pull centos/mongodb-32-centos7
$ docker pull centos/mongodb-34-centos7
```

要使用这些镜像，您可直接从这些 registry 访问镜像或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 ImageStream。您可以找到所有提供的 OpenShift Container Platform 镜像的 ImageStream 定义 示例。

3.4.4. 配置和使用

3.4.4.1. 初始化数据库

您可以使用一个临时卷或持久性卷配置 MongoDB。您第一次使用卷时，与数据库管理员用户一起创建数据库。随后会启动 MongoDB 守护进程。如果您要将卷重新关联到另一个容器，则不会创建数据库、数据库用户和管理员用户，并且会启动 MongoDB 守护进程。

以下命令创建一个带有临时卷在容器中运行的 MongoDB 的新数据库 pod:

```
$ oc new-app \
    -e MONGODB_USER=<username> \
    -e MONGODB_PASSWORD=<password> \
    -e MONGODB_DATABASE=<database_name> \
    -e MONGODB_ADMIN_PASSWORD=<admin_password> \
    registry.redhat.io/rhscl/mongodb-26-rhel7
```

3.4.4.2. 在容器中运行 MongoDB 命令

OpenShift Container Platform 使用 Software Collections (SCLs)来安装和启动 MongoDB。如果要在运行的容器内执行 MongoDB 命令（用于调试）,则必须使用 bash 调用它。

要做到这一点，首先请确定正在运行的 MongoDB Pod 的名称。例如，您可以查看当前项目中的 pod 列表：

```
$ oc get pods
```

然后，打开到所需 pod 的远程 shell 会话：

```
$ oc rsh <pod>
```

在进入容器时，会自动启用所需的 SCL。

现在，您可以在 bash shell 中运行 **mongo** 命令来启动 MongoDB 互动会话并执行普通的 MongoDB 操作。例如：要切换到 **sampledb** 数据库并验证为数据库用户：

```
bash-4.2$ mongo -u $MONGODB_USER -p $MONGODB_PASSWORD $MONGODB_DATABASE
MongoDB shell version: 2.6.9
connecting to: sampledb
>
```

完成后，按 **CTRL+D** 退出 MongoDB 会话。

3.4.4.3. 环境变量

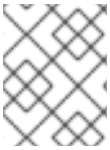
MongoDB 用户名、密码、数据库名称和 **admin** 密码必须使用以下环境变量进行配置：

表 3.5. MongoDB 环境变量

变量名称	描述
MONGODB_USER	要创建的 MongoDB 帐户的用户名。
MONGODB_PASSWORD	用户帐户的密码。
MONGODB_DATABASE	数据库名称。
MONGODB_ADMIN_PASSWORD	**admin** 用户的密码。

警告

您必须指定用户名、密码、数据库名称和 **admin** 密码。如果没有指定全部 4 个，pod 将无法启动，OpenShift Container Platform 会不断尝试重启它。



注意

管理员的用户名被设置为 **admin**，您必须通过设置 **MONGODB_ADMIN_PASSWORD** 环境变量来指定它的密码。这个过程是在数据库初始化时完成的。

MongoDB 设置可使用以下环境变量进行配置：

表 3.6. 额外的 MongoDB 设置

变量名称	描述	默认
MONGODB_NOPREALLOC	禁用数据文件预分配。	**true**
MONGODB_SMALLFILES	将 MongoDB 设置为使用较小的默认数据文件大小。	**true**
MONGODB_QUIET	以静默模式运行 MongoDB 试图限制输出量。	**true**

注意

在 MongoDB 版本 2.6 及更高版本中默认启用文本搜索，因此没有可配置的参数。

3.4.4.4. 卷挂载点

MongoDB 镜像可以使用挂载的卷运行，以便为数据库启用持久性存储：

- */var/lib/mongodb/data* - 这是 MongoDB 存储数据库文件的数据库目录。

3.4.4.5. 更改密码

密码是镜像配置的一部分，因此唯一支持的为数据库用户（**MONGODB_USER**）和 admin 用户更改密码的方法是分别更改环境变量 **MONGODB_PASSWORD** 和 **MONGODB_ADMIN_PASSWORD**。

您可以通过在 web 控制台中查看 pod 或部署配置，或通过 CLI 列出环境变量来查看当前的密码：

```
$ oc set env pod <pod_name> --list
```

直接在 MongoDB 中更改数据库密码会导致变量中保存的值与实际密码不匹配。每当数据库容器启动时，它会将密码重置至环境变量中存储的值。

要更改这些密码，使用 **oc set env** 命令为相关的部署配置更新一个或多个所需的环境变量。如果多个部署配置使用这些环境变量，例如从模板创建应用程序时，您必须更新每个部署配置上的变量，以便密码可在所有方面同步。这可以在同一命令中完成：

```
$ oc set env dc <dc_name> [<dc_name_2> ...] \
  MONGODB_PASSWORD=<new_password> \
  MONGODB_ADMIN_PASSWORD=<new_admin_password>
```

重要

根据您的应用程序，应用程序的其它部分可能还会有其他密码的环境变量，它们也应该更新以匹配。例如，前端 pod 中可能会有一个更通用的 **DATABASE_USER** 变量，该变量应该与数据库用户的密码匹配。确保每个应用程序正在同步所有必需的环境变量，否则您的 pod 可能无法在触发时重新部署。

如果您 配置更改触发器，更新环境变量会触发数据库服务器的重新部署。否则，您必须手动启动新的部署以应用密码更改。

要验证新密码是否生效，首先打开正在运行的 MongoDB pod 的远程 shell 会话：

```
$ oc rsh <pod>
```

在 bash shell 中验证数据库用户的新密码：

```
bash-4.2$ mongo -u $MONGODB_USER -p <new_password> $MONGODB_DATABASE --eval
"db.version()"
```

如果正确修改了密码，您应该看到类似如下的输出：

```
MongoDB shell version: 2.6.9
connecting to: sampledb
2.6.9
```

验证 **admin** 用户的新密码：

```
bash-4.2$ mongo -u admin -p <new_admin_password> admin --eval "db.version()"
```

如果正确修改了密码，您应该看到类似如下的输出：

```
-
```

```
MongoDB shell version: 2.6.9
connecting to: admin
2.6.9
```

3.4.5. 从模板创建数据库服务

OpenShift Container Platform 提供了一个 模板，可简化新数据库服务的创建。模板提供参数字段来定义所有强制环境变量（用户、密码、数据库名称等）,并使用预先定义的默认值（包括密码值的自动生成）。它还将定义 部署配置 和 服务。

在初始集群设置过程中，您的集群管理员应该在默认 **openshift** 项目中注册 MongoDB 模板。如需了解更多详细信息，请参阅载入默认镜像流和模板。

有两个可用模板：

- **Mongodb-ephemeral** 只是用于开发/测试目的，因为它对数据库内容使用临时存储。这意味着，如果数据库 pod 因某种原因被重启，如 pod 正在移至另一节点，或正在更新和触发重新部署的部署配置，则所有数据将会丢失。
- **Mongodb-persistent** 使用持久性卷存储来保存数据库数据，这意味着 pod 重启后数据会保留下来。使用持久性卷需要在 OpenShift Container Platform 部署中定义持久性卷池。设置池的信息包括在使用 NFS 的持久性存储 中。

您可以按照以下步骤实例化模板 https://access.redhat.com/documentation/en-us/openshift_container_platform/3.11/html-single/developer_guide/#dev-guide-templates。

在实例化该服务后，您可以将用户名、密码和数据库名称环境变量复制到旨在访问数据库的另一个组件的部署配置中。然后该组件可以通过定义的服务访问数据库。

3.4.6. MongoDB 复制

注意

作为示例提供了为数据库镜像启用集群的配置，它们不适用于生产环境。

红帽使用 StatefulSet 为 MongoDB 复制（集群）提供了概念验证 模板。您可以 从 GitHub 获取示例模板。

例如，将示例模板上传到当前项目的模板库中：

```
$ oc create -f \
    https://raw.githubusercontent.com/sclorg/mongodb-container/master/examples/petset/mongodb-petset-persistent.yaml
```

重要

示例模板使用持久性存储。集群中必须有持久性卷才能使用此模板。

因为 OpenShift Container Platform 会自动重启不健康的 Pod（容器），如果一个或多个成员崩溃或失败，它将重启副本设置成员。

虽然副本设置的成员被缩减或正在重启，但可能是以下情况之一：

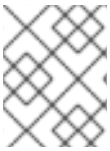
1. 主要成员为下：
   在这种情况下，其他两个成员选择一个新的 PRIMARY。直到那个时候，读取不会受到影响，但写入操作会失败。成功选举后，通常会写入并读取过程。

2. SECONDARY 的一个成员会停止：
   读取和写入不会受到影响。根据 **oplogSize** 配置和写入率，第三方可能无法加入副本集，需要手动干预来重新同步其数据库的副本。

3. 任何两个成员都处于停止状态：
   当一个由三个成员组成的副本集的成员无法访问其他成员时，如果它是 PRIMARY，则会从这个角色下移。在这种情况下，SECONDARY 成员可能会提供 read 功能，写入操作会失败。一旦有多个成员可用，会选举一个新的 PRIMARY 成员，读写操作可正常进行。

4. 所有成员都停机：
   在这种极端情况下，读取和写入都失败。在两个或者两个以上成员被备份后，选举会重新建立副本集，使其具有 PRIMARY 和 SECONDARY 成员，之后读取和写入将可以正常工作。

这是推荐的 MongoDB 复制策略。



注意

对于生产环境，您必须尽可能在成员间保持分离。建议您使用一个或多个节点选择功能将 StatefulSet pod 调度到不同的节点上，并提供由独立卷支持的存储。

3.4.6.1. 限制：

- 只支持 MongoDB 3.2。
- 在进行缩减时，您必须手动更新副本设置的配置。
- 更改用户和管理员密码是一个手动过程。它要求：
  - 更新 StatefulSet 配置中的环境变量值,
  - 更改数据库中的密码，并
  - 重启所有 pod。

3.4.6.2. 使用示例模板

假设您已经有三个预先创建的持久性卷或配置了持久性卷置备。

1. 创建一个新的项目，其中要创建一个 MongoDB 集群：

```
$ oc new-project mongodb-cluster-example
```

2. 使用示例模板创建新应用程序：

```
$ oc new-app https://raw.githubusercontent.com/sclorg/mongodb-
container/master/examples/petset/mongodb-petset-persistent.yaml
```

   此命令创建带有三个副本设置成员的 MongoDB 集群。

3. 检查新的 MongoDB Pod 的状态：

```
$ oc get pods
NAME        READY     STATUS    RESTARTS   AGE
mongodb-0   1/1       Running   0          50s
mongodb-1   1/1       Running   0          50s
mongodb-2   1/1       Running   0          49s
```

从示例模板创建集群后，您就有 3 个成员设置了副本。运行 pod 后，您可以对这些 pod 执行各种操作，例如：

- 检查其中一个 pod 的日志：

```
$ oc logs mongodb-0
```

- 登录到 pod:

```
$ oc rsh mongodb-0
sh-4.2$
```

- 登录到 MongoDB 实例：

```
sh-4.2$ mongo $MONGODB_DATABASE -u $MONGODB_USER -
p$MONGODB_PASSWORD
MongoDB shell version: 3.2.6
connecting to: sampledb
rs0:PRIMARY>
```

3.4.6.3. 扩展

MongoDB 建议副本集中的成员数量是单数。如果有足够的持久性卷，或者存在动态存储置备程序，则使用 **oc scale** 命令进行扩展：

```
$ oc scale --replicas=5 statefulsets/mongodb

$ oc get pods
NAME        READY     STATUS    RESTARTS   AGE
mongodb-0   1/1       Running   0          9m
mongodb-1   1/1       Running   0          8m
mongodb-2   1/1       Running   0          8m
mongodb-3   1/1       Running   0          1m
mongodb-4   1/1       Running   0          57s
```

这会创建新的 pod 来连接副本集并更新其配置。

注意

如果数据库大小大于 **oplogSize** 配置，则扩展现有数据库需要人工干预。对于这样的情况，需要手动对新成员进行初始同步。如需更多信息，请参阅 检查 Oplog 的大小 和 MongoDB Replication 文档。

3.4.6.4. 缩减

若要缩减副本集，可以将成员从 5 个缩减到 3 个，或从 3 个缩减到 1 个。

虽然当满足条件（存储可用性、现有数据库的大小和 **oplogSize**）时，可以在不需要人工干预的情况下进行扩展，但缩减总是需要人工干预。

缩减：

1. 使用 **oc scale** 命令设置新副本数：

```
$ oc scale --replicas=3 statefulsets/mongodb
```

如果新副本数仍为前一个数字中的大多数，则副本集可能会选择一个新的 PRIMARY，如果删除的 pod 中的一个具有 PRIMARY 成员角色。例如，从 5 个成员缩减到 3 个成员时。

或者，缩减到较低数字可临时导致副本集只有 SECONDARY 成员，且处于只读模式。例如，从 5 个成员缩减到只有 1 个成员时。

2. 更新 replica set 配置，以删除不存在的成员。
这在以后可能会有所改进，一个可能的实现是设置 **PreStop** pod hook 来检查副本数（通过 Downward API 获得），并确定 pod 已从 StatefulSet 中删除，并且由于其他原因不会重启。

3. 清除已弃用 pod 使用的卷。

3.5. MARIADB

3.5.1. 概述

OpenShift Container Platform 为运行 MariaDB 提供容器镜像。该镜像可根据配置文件中提供的用户名、密码和数据库名称设置提供数据库服务。

3.5.2. 版本

目前，OpenShift Container Platform 提供了 MariaDB 的 10.0 和 10.1 版本。

3.5.3. 镜像

根据您的需要，这些镜像分为两种类型：

- RHEL 7
- CentOS 7

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/rhscl/mariadb-100-rhel7
$ docker pull registry.redhat.io/rhscl/mariadb-101-rhel7
```

基于 CentOS 7 的镜像

这些镜像在 Docker Hub 上可用：

```
$ docker pull openshift/mariadb-100-centos7
$ docker pull centos/mariadb-101-centos7
```

要使用这些镜像，您可直接从这些 registry 访问镜像或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 ImageStream。您可以找到所有提供的 OpenShift Container Platform 镜像的 ImageStream 定义 示例。

3.5.4. 配置和使用

3.5.4.1. 初始化数据库

您第一次使用共享卷时，数据库会与数据库管理员用户和 MariaDB root 用户一起创建（如果您指定了 **MYSQL_ROOT_PASSWORD** 环境变量）。随后会启动 MariaDB 守护进程。如果您要将卷重新关联到另一个容器，则不会创建数据库、数据库用户和管理员用户，并且会启动 MariaDB 守护进程。

以下命令创建一个在容器中运行的 MariaDB 的新数据库 pod:

```
$ oc new-app \
    -e MYSQL_USER=<username> \
    -e MYSQL_PASSWORD=<password> \
    -e MYSQL_DATABASE=<database_name> \
    registry.redhat.io/rhscl/mariadb-101-rhel7
```

3.5.4.2. 在容器中运行 MariaDB 命令

OpenShift Container Platform 使用 Software Collections (SCLs)来安装和启动 MariaDB。如果要在运行的容器内执行 MariaDB 命令（用于调试）,则必须使用 bash 调用它。

要做到这一点，首先确定正在运行的 MariaDB Pod 的名称。例如，您可以查看当前项目中的 pod 列表：

```
$ oc get pods
```

然后，打开到 pod 的远程 shell 会话：

```
$ oc rsh <pod>
```

在进入容器时，会自动启用所需的 SCL。

现在，您可以在 bash shell 中运行 **mysql** 命令来启动 MariaDB 互动会话并执行普通的 MariaDB 操作。例如，验证数据库用户：

```
bash-4.2$ mysql -u $MYSQL_USER -p$MYSQL_PASSWORD -h $HOSTNAME
$MYSQL_DATABASE
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.5.37 MySQL Community Server (GPL)
...
mysql>
```

完成后，输入 **quit** 或 **exit** 退出 MySQL 会话。

3.5.4.3. 环境变量

MariaDB 用户名、密码和数据库名称必须使用以下环境变量进行配置：

表 3.7. MariaDB 环境变量

变量名称	描述
MYSQL_USER	要创建的 MySQL 帐户的用户名。
MYSQL_PASSWORD	用户帐户的密码。
MYSQL_DATABASE	数据库名称。
MYSQL_ROOT_PASSWORD	root 用户的密码（可选）。

警告

您必须指定用户名、密码和数据库名称。如果没有指定全部三个变量，pod 将无法启动，OpenShift Container Platform 会不断尝试重启它。

MariaDB 设置可使用以下环境变量进行配置：

表 3.8. 其它 MariaDB 设置

变量名称	描述	默认
MYSQL_LOWER_CASE_TABLE_NAMES	设定表名称的存储和比较方式。	0
MYSQL_MAX_CONNECTIONS	允许客户端同时连接的最大数量。	151
MYSQL_MAX_ALLOWED_PACKET	一个数据包或生成/中间字符串的最大值。	200M
MYSQL_FT_MIN_WORD_LEN	FULLTEXT 索引中包含的单词的最小长度。	4
MYSQL_FT_MAX_WORD_LEN	FULLTEXT 索引中包含的单词的最大长度。	20
MYSQL_AIO	如果原生 AIO 出行问题，则控制 **innodb_use_native_aio** 设置值。	1
MYSQL_TABLE_OPEN_CACHE	所有线程打开的表的数量。	400

变量名称	描述	默认
MYSQL_KEY_BUFFER_SIZE	用于索引块的缓冲区的大小。	32M（或者 10% 可用内存）
MYSQL_SORT_BUFFER_SIZE	用于排序的缓冲区的大小。	256K
MYSQL_READ_BUFFER_SIZE	用于后续扫描的缓冲区的大小。	8M（或者 5% 可用内存）
MYSQL_INNODB_BUFFER_POOL_SIZE	InnoDB 缓存表和索引数据的缓冲池的大小。	32M（或者 50% 可用内存）
MYSQL_INNODB_LOG_FILE_SIZE	日志组中每个日志文件的大小。	8M（或者 15% 可用内存）
MYSQL_INNODB_LOG_BUFFER_SIZE	InnoDB 用来写入磁盘日志文件的缓冲大小。	8M（或者 15% 可用内存）
MYSQL_DEFAULTS_FILE	指向其它配置文件。	/etc/my.cnf
MYSQL_BINLOG_FORMAT	设置 binlog 格式，支持的值是 **row** 和 **statement**。	声明

3.5.4.4. 卷挂载点

MariaDB 镜像可以使用挂载的卷运行，以便为数据库启用持久性存储：

- */var/lib/mysql/data* -MariaDB 存储数据库文件的 MySQL 数据目录。

注意

将目录从主机挂载到容器时，请确保挂载的目录具有适当的权限。同时还要验证目录的拥有者和组是否与容器中运行的用户名匹配。

3.5.4.5. 更改密码

密码是镜像配置的一部分，因此唯一支持的为数据库用户（**MYSQL_USER**）和 admin 用户更改密码的方法是分别更改环境变量 **MYSQL_PASSWORD** 和 **MYSQL_ROOT_PASSWORD**。

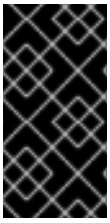
您可以通过在 web 控制台中查看 pod 或部署配置，或通过 CLI 列出环境变量来查看当前的密码：

```
$ oc set env pod <pod_name> --list
```

通过 SQL 语句或者通过环境变量更改数据库密码会导致变量中保存的值与实际密码不匹配。每当数据库容器启动时，它会将密码重置至环境变量中存储的值。

要更改这些密码，使用 **oc set env** 命令为相关的部署配置更新一个或多个所需的环境变量。如果多个部署配置使用这些环境变量，例如从模板创建应用程序时，您必须更新每个部署配置上的变量，以便密码可在所有方面同步。这可以在同一命令中完成：

```
$ oc set env dc <dc_name> [<dc_name_2> ...] \
  MYSQL_PASSWORD=<new_password> \
  MYSQL_ROOT_PASSWORD=<new_root_password>
```



重要

根据您的应用程序，应用程序的其它部分可能还会有其他密码的环境变量，它们也应该更新以匹配。例如，前端 pod 中可能会有一个更通用的 **DATABASE_USER** 变量，该变量应该与数据库用户的密码匹配。确保每个应用程序正在同步所有必需的环境变量，否则您的 pod 可能无法在触发时重新部署。

如果您 配置更改触发器，更新环境变量会触发数据库服务器的重新部署。否则，您必须手动启动新的部署以应用密码更改。

要验证新密码是否生效，首先请打开正在运行的 MariaDB pod 的远程 shell 会话：

```
$ oc rsh <pod>
```

在 bash shell 中验证数据库用户的新密码：

```
bash-4.2$ mysql -u $MYSQL_USER -p<new_password> -h $HOSTNAME $MYSQL_DATABASE -te
"SELECT * FROM (SELECT database()) db CROSS JOIN (SELECT user()) u"
```

如果正确修改了密码，您应该可以看到类似如下的表：

```
+------------+--------------------+
| database() | user()             |
+------------+--------------------+
| sampledb   | user0PG@172.17.42.1 |
+------------+--------------------+
```

验证 **root** 用户的新密码：

```
bash-4.2$ mysql -u root -p<new_root_password> -h $HOSTNAME $MYSQL_DATABASE -te
"SELECT * FROM (SELECT database()) db CROSS JOIN (SELECT user()) u"
```

如果正确修改了密码，您应该可以看到类似如下的表：

```
+------------+-----------------+
| database() | user()          |
+------------+-----------------+
| sampledb   | root@172.17.42.1 |
+------------+-----------------+
```

3.5.5. 从模板创建数据库服务

OpenShift Container Platform 提供了一个 模板，可简化新数据库服务的创建。模板提供参数字段来定义所有强制环境变量（用户、密码、数据库名称等）,并使用预先定义的默认值（包括密码值的自动生成）。它还将定义 部署配置 和 服务。

在初始集群设置过程中，集群管理员应该在默认 **openshift** 项目中注册 MariaDB 模板。如需了解更多详细信息，请参阅载入默认镜像流和模板。

有两个可用模板：

- **Mariadb-ephemeral** 仅用于开发或测试目的，因为它对数据库内容使用临时存储。这意味着，如果数据库 pod 因某种原因被重启，如 pod 正在移至另一节点，或正在更新和触发重新部署的部署配置，则所有数据将会丢失。
- **Mariadb-persistent** 使用持久性卷存储来保存数据库数据，这意味着 pod 重启后数据会保留下来。使用持久性卷需要在 OpenShift Container Platform 部署中定义持久性卷池。设置池的信息包括在使用 NFS 的持久性存储 中。

您可以按照以下步骤实例化模板 https://access.redhat.com/documentation/en-us/openshift_container_platform/3.11/html-single/developer_guide/#dev-guide-templates。

在实例化该服务后，您可以将用户名、密码和数据库名称环境变量复制到旨在访问数据库的另一个组件的部署配置中。然后该组件可以通过定义的服务访问数据库。

3.5.6. 故障排除

本节描述了您可能会遇到的问题，并给出可能的解决方案。

3.5.6.1. Linux Native AIO 失败

症状

MySQL 容器无法启动，日志显示类似如下：

```
151113  5:06:56 InnoDB: Using Linux native AIO
151113  5:06:56  InnoDB: Warning: io_setup() failed with EAGAIN. Will make 5 attempts before
giving up.
InnoDB: Warning: io_setup() attempt 1 failed.
InnoDB: Warning: io_setup() attempt 2 failed.
Waiting for MySQL to start ...
InnoDB: Warning: io_setup() attempt 3 failed.
InnoDB: Warning: io_setup() attempt 4 failed.
Waiting for MySQL to start ...
InnoDB: Warning: io_setup() attempt 5 failed.
151113  5:06:59  InnoDB: Error: io_setup() failed with EAGAIN after 5 attempts.
InnoDB: You can disable Linux Native AIO by setting innodb_use_native_aio = 0 in my.cnf
151113  5:06:59 InnoDB: Fatal error: cannot initialize AIO sub-system
151113  5:06:59 [ERROR] Plugin 'InnoDB' init function returned error.
```

```
151113  5:06:59 [ERROR] Plugin 'InnoDB' registration as a STORAGE ENGINE failed.
151113  5:06:59 [ERROR] Unknown/unsupported storage engine: InnoDB
151113  5:06:59 [ERROR] Aborting
```

解释

由于资源限制，MariaDB 的存储引擎无法使用内核的 AIO（异步 I/O）功能。

解决方案

通过将环境变量 **MYSQL_AIO** 设置为 **0** 来完全关闭 AIO使用。在后续的部署中，这会让 MySQL 配置变量 **innodb_use_native_aio** 的值为 **0**。

另外，还可增加 **aio-max-nr** 内核资源。下面的例子检查了 **aio-max-nr** 的当前值并加倍。

```
$ sysctl fs.aio-max-nr
fs.aio-max-nr = 1048576
# sysctl -w fs.aio-max-nr=2097152
```

这会针对每个节点进行解析，持续到下一个节点重启为止。

第 4 章 其他镜像

4.1. 概述

本节组包含可供 OpenShift Container Platform 用户使用的其他容器镜像的信息。

4.2. JENKINS

4.2.1. 概述

OpenShift Container Platform 为运行 Jenkins 提供容器镜像。此镜像提供 Jenkins 服务器实例，可用于为连续测试、集成和交付设置基本流程。

此镜像还包括一个 Jenkins 任务示例，用于触发 OpenShift Container Platform 中定义的 **BuildConfig** 的新构建，测试该构建的输出，然后在成功构建时重新标记构建已准备好生产环境。如需了解更多详细信息，请参阅 README。

OpenShift Container Platform 遵从 Jenkins 的 LTS 的发行版本。OpenShift Container Platform 提供了一个包含 Jenkins 2.x 的镜像。以前会提供一个带有 Jenkins 1.x 的独立镜像，但现在不再维护。

4.2.2. 镜像

OpenShift Container Platform Jenkins 镜像采用两种类型：

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/openshift3/jenkins-2-rhel7
```

基于 CentOS 7 的镜像

此镜像在 Docker Hub 上可用：

```
$ docker pull openshift/jenkins-2-centos7
```

要使用这些镜像，您可直接从这些 registry 访问镜像或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 ImageStream。您可以找到所有提供的 OpenShift Container Platform 镜像的 ImageStream 定义示例。

4.2.3. 配置和自定义

4.2.3.1. 身份验证

您可采用两种方式管理 Jenkins 身份验证：

- 由 OpenShift Login 插件提供的 OpenShift Container Platform Oauth 身份验证。
- 由 Jenkins 提供的标准身份验证。

4.2.3.1.1. OpenShift Container Platform OAuth 身份验证

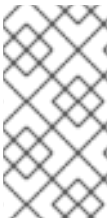
OAuth 身份验证 通过在 Jenkins UI 中 **配置 Configure Global Security** 面板，或者将 Jenkins **Deployment Config** 上的 **OPENSHIFT_ENABLE_OAUTH** 环境变量设置为非 **false** 来激活。这会激活 OpenShift Login 插件，该插件从 pod 数据或通过与 OpenShift Container Platform API 服务器交互来检索配置信息。

有效凭证由 OpenShift Container Platform 身份提供程序控制。例如，如果 **Allow All** 是默认身份提供程序，您可以为用户名和密码提供任何非空字符串。

Jenkins 支持浏览器和非浏览器访问。

登录时，有效用户会自动添加到 Jenkins 授权列表中，其中的 OpenShift Container Platform **Role** 规定了用户拥有的特定 Jenkins 权限。

具有 **Admin** 角色的用户拥有传统 Jenkins 管理用户权限，具有 **edit** 或 **view** 角色的用户的权限会逐渐减少。如需 OpenShift 角色到 Jenkins 权限映射的特定信息，请参阅 Jenkins 镜像源存储库 README。



注意

使用 OpenShift Container Platform OAuth 时，OpenShift Container Platform Jenkins 镜像中预填充了管理特权的 **admin** 用户不会被授予这些权限，除非 OpenShift Container Platform 集群管理员在 OpenShift Container Platform 身份提供程序中明确定义了该用户，并将 **admin** 角色分配给用户。

最初创建用户后，可以更改 Jenkins 用户权限。OpenShift Login 插件轮询 OpenShift Container Platform API 服务器以获取权限，并利用从 OpenShift Container Platform 检索的权限更新存储在 Jenkins 中的每个用户的权限。如果 Jenkins UI 用于为 Jenkins 用户更新权限，则权限更改将在插件下次轮询 OpenShift Container Platform 时被覆盖。

您可通过 **OPENSHIFT_permissions_poll_interval** 环境变量来控制轮询频率。默认轮询间隔为五分钟。

使用 OAuth 身份验证创建新 Jenkins 服务的最简单方法是 使用模板，如下所述。

4.2.3.1.2. Jenkins 标准身份验证

如果镜像未使用模板直接运行，则默认使用 Jenkins 身份验证。

Jenkins 首次启动时，配置与管理员用户和密码一同创建。默认用户凭证为 **admin** 和 **password**。在使用标准 Jenkins 身份验证时，且仅这种情况下，通过设置 **JENKINS_PASSWORD** 环境变量来配置默认密码。

使用标准 Jenkins 身份验证创建新 Jenkins 应用程序：

```
$ oc new-app -e \
    JENKINS_PASSWORD=<password> \
    openshift/jenkins-2-centos7
```

4.2.3.2. 环境变量

Jenkins 服务器可通过以下环境变量进行配置：

- **OPENSHIFT_ENABLE_OAUTH**（默认：**false**）
  决定在登录 Jenkins 时，OpenShift Login 插件可否管理身份验证。要启用，请设为 **true**。
- **JENKINS_PASSWORD**（默认： **password**）

使用标准 Jenkins 身份验证时 **admin** 用户的密码。**OPENSHIFT_ENABLE_OAUTH** 设置为 **true** 时不适用。

- **OPENSHIFT_JENKINS_JVM_ARCH**
  设置为 **x86_64** 或 **i386**，以覆盖用于托管 Jenkins 的 JVM。为提高内存效率，如果容器中运行的内存限值为 2GiB，则默认情况下 Jenkins 镜像会动态使用 32 位 JVM。

- **JAVAINER_MAX_HEAP_PARAM**
  CONTAINER_HEAP_PERCENT（默认值： **0.5** 或 50%）
  JENKINS_MAX_HEAP_UPPER_BOUND_MB
  这些值控制 Jenkins JVM 的最大堆大小。如果设置了 **JAVA_MAX_HEAP_PARAM**（示例设置： **-Xmx512m**），则优先使用其值。否则，最大堆大小将动态计算为容器内存限值的 **CONTAINER_HEAP_PERCENT**%（示例设置： **0.5** 或 50%），可选上限为 **JENKINS_MAX_HEAP_UPPER_BOUND_MB** MiB（示例设置： **512**）。

  默认情况下，Jenkins JVM 的最大堆大小设置为容器内存限值的 50%，且无上限。

- **JAVA_INITIAL_HEAP_PARAM**
  CONTAINER_INITIAL_PERCENT
  这些值控制 Jenkins JVM 的初始堆大小。如果设置了 **JAVA_INITIAL_HEAP_PARAM**（示例设置： **-Xms32m**），则优先使用其值。否则，初始堆大小可能会动态计算为 **CONTAINER_INITIAL_PERCENT**%（示例： **0.1** 或 10%）动态计算的最大堆大小。

  默认情况下，初始堆大小留给 JVM。

- **CONTAINER_CORE_LIMIT**
  如果设置，请将用于调整内部 JVM 线程数的内核数指定为整数。示例设置：**2**。

- **JAVA_TOOL_OPTIONS** (default: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**)
  指定此容器中运行的所有 JVM 需要注意的选项。不建议覆盖此选项。

- **JAVA_GC_OPTS** (default: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaiveSizePolicyWeight=90**)
  指定 Jenkins JVM 垃圾回收参数。不建议覆盖此选项。

- **JENKINS_JAVA_OVERRIDES**
  指定适用于 Jenkins JVM 的附加选项。这些选项附加到所有其他选项中，包括上面的 Java 选项，必要时可用于覆盖其中任何一个选项。用空格分开各个附加选项；如有任意选项包含空格字符，请使用反斜杠转义。示例设置：**-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value**。

- **JENKINS_OPTS**
  为 Jenkins 指定参数。

- **INSTALL_PLUGINS**
  指定在容器首次运行或 **OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS** 设置为 **true** 时需要安装的 Jenkins 附加插件。（参看以下内容）插件被指定为用逗号分隔的"名称:版本"对列表。示例设置：**git:3.7.0,subversion:2.10.2**。

- **OPENSHIFT_PERMISSIONS_POLL_INTERVAL**（默认：**300000** - 5 分钟）
  以毫秒为单位指定 OpenShift Login 插件轮询 OpenShift Container Platform 的频率，以毫秒为单位指定与 Jenkins 中定义的每个用户关联的权限。

- **OVERRIDE_PV_CONFIG_WITH_IMAGE_CONFIG**（默认： **false**）
  当为 Jenkins 配置目录运行带有一个 OpenShift Container Platform 持久性卷运行此镜像时，从镜像到持久性卷的配置传输仅进行镜像的第一个启动，因为持久性卷由持久性卷声明创建来分配。如果您在初始启动后创建自定义镜像来扩展此镜像并更新自定义镜像中的配置，则默认不会

复制它，除非将该环境变量设置为 **true**。

- **OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS**（默认： **false**）
  当使用 Jenkins 配置目录的 OpenShift Container Platform 持久性卷运行此镜像时，从镜像到持久性卷的插件传输仅进行镜像的第一个启动，因为持久性卷由持久性卷声明创建来分配。如果您在初始启动后创建可扩展此镜像的自定义镜像并更新自定义镜像中的插件，则默认不会复制它们，除非将该环境变量设置为 **true**。

- **ENABLE_FATAL_ERROR_LOG_FILE**（默认： **false**）
  当使用 Jenkins 配置目录的 OpenShift Container Platform 持久性声明运行此镜像时，该环境变量允许在严重错误发生时，严重错误日志文件保留。严重错误文件保存在：**/var/lib/jenkins/logs**。

- **NODEJS_SLAVE_IMAGE**
  设置此值将覆盖用于默认 NodeJS 代理 Pod 配置的镜像。默认 NodeJS 代理 pod 使用 **docker.io/openshift/jenkins-agent-nodejs-8-centos7** 或 **registry.redhat.io/openshift3/jenkins-agent-nodejs-8-rhel7**，具体取决于您运行 Jenkins 镜像的 CentOS 或 RHEL 版本。该变量必须在 Jenkins 首次启动前进行设置，以便其生效。

- **MAVEN_SLAVE_IMAGE**
  设置此值将覆盖用于默认 Maven 代理 Pod 配置的镜像。默认 maven 代理 pod 使用 **docker.io/openshift/jenkins-agent-maven-35-centos7** 或 **registry.redhat.io/openshift3/jenkins-agent-maven-35-rhel7**，具体取决于您运行 Jenkins 镜像的 CentOS 或 RHEL 版本。该变量必须在 Jenkins 首次启动前进行设置，以便其生效。

- **JENKINS_UC_INSECURE**
  确定在 Jenkins Update Center 存储库使用无效的 SSL 证书时是否允许 Jenkins 插件下载。如果使用了带有未知 CA 的自签名证书，或者在中间使用了一个企业级代理。此变量适用于插件下载，这些插件下载可能会在 Jenkins 镜像构建过程中发生，或构建了 Jenkins 镜像的扩展。当运行 Jenkins 镜像并使用其中一个选项下载额外插件时，还会应用它，包括带有 plugins.txt 的 S2I 或 INSTALL_PLUGINS 环境变量。设置为 true 以启用此变量。

4.2.3.3. 跨项目访问

如果您要作为同一项目内的部署在其它位置运行 Jenkins，则需要为 Jenkins 提供访问令牌来访问您的项目。

1. 识别具有适当权限访问 Jenkins 需要访问项目的服务帐户的 secret:

```
$ oc describe serviceaccount jenkins
Name:       default
Labels:     <none>
Secrets:    {  jenkins-token-uyswp    }
            {  jenkins-dockercfg-xcr3d    }
Tokens:     jenkins-token-izv1u
            jenkins-token-uyswp
```

   这种情况下，secret 被命名为 **jenkins-token-extensionswp**。

2. 从 secret 中检索令牌：

```
$ oc describe secret <secret name from above> # for example, jenkins-token-uyswp
Name:       jenkins-token-uyswp
Labels:     <none>
Annotations:    kubernetes.io/service-account.name=jenkins,kubernetes.io/service-
account.uid=32f5b661-2a8f-11e5-9528-3c970e3bf0b7
```

```
Type:   kubernetes.io/service-account-token
Data
====
ca.crt: 1066 bytes
token:  eyJhbGc..<content cut>....wRA
```

令牌字段包含 Jenkins 访问项目所需的令牌值。

4.2.3.4. 卷挂载点

可使用挂载卷运行 Jenkins 镜像，以便为配置启用持久性存储：

- */var/lib/jenkins* - 这是 Jenkins 存储配置文件的数据目录，包含任务定义。

4.2.3.5. 通过 Source-To-Image 自定义 Jenkins 镜像

要自定义官方 OpenShift Container Platform Jenkins 镜像，有两个选项：

- 使用 Docker 层。
- 使用镜像作为 Source-To-Image 构建器，如下所述。

您可使用 S2I 来复制自定义 Jenkins 任务定义，添加其它插件，或使用您自己的自定义配置来替换所提供的 *config.xml* 文件。

要在 Jenkins 镜像中包括您的修改，您需要具有以下目录结构的 Git 存储库：

plugins
该目录包含要复制到 Jenkins 中的二进制 Jenkins 插件。

plugins.txt
此文件列出了您要安装的插件：

```
pluginId:pluginVersion
```

configuration/jobs
该目录包含 Jenkins 任务定义。

configuration/config.xml
该文件包含您的自定义 Jenkins 配置。

configuration/ 目录的内容将复制到 */var/lib/jenkins/* 目录中，以便还可以包含其他文件，比如 *credentials.xml*。

以下是在 OpenShift Container Platform 中自定义 Jenkins 镜像的示例构建配置：

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source:                    1
    git:
      uri: https://github.com/custom/repository
    type: Git
```

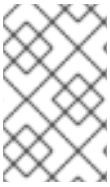
```
strategy:                    2
  sourceStrategy:
    from:
      kind: ImageStreamTag
      name: jenkins:latest
      namespace: openshift
  type: Source
output:                      3
  to:
    kind: ImageStreamTag
    name: custom-jenkins:latest
```

1. **source** 字段使用上述布局定义源 Git 存储库。
2. **strategy** 字段定义用作构建的源镜像的原始 Jenkins 镜像。
3. **output** 字段定义了可在部署配置中使用的生成的自定义 Jenkins 镜像，而不是官方 Jenkins 镜像。

4.2.3.6. 配置 Jenkins Kubernetes 插件

OpenShift Container Platform Jenkins 镜像包含预装的 Kubernetes 插件，支持使用 Kubernes 和 OpenShift Container Platform 在多个容器主机上动态置备 Jenkins 代理。

为了使用 Kubernetes 插件，OpenShift Container Platform 提供了适合用作 Jenkins 代理的五个镜像：***Base***、***Maven*** 和 ***Node.js*** 镜像。如需更多信息，请参阅 Jenkins 代理。



注意

jenkins-slave-maven-* 和 jenkins-slave-nodejs-* 镜像在 v3.10 发行周期中被标记为已弃用。临时镜像仍会存在，因此用户可以将其应用程序迁移到较新的 jenkins-agent-maven-* 和 jenkins-agent-nodejs-* 镜像中。

Maven 和 Node.js 代理镜像均会在 OpenShift Container Platform Jenkins 镜像的 Kubernetes 插件配置中自动配置为 Kubernetes Pod 模板镜像。该配置包含各个镜像的标签，可应用于 “Restrict where this project can be run” 设置下的任何 Jenkins 任务。如果应用了标签，则指定任务的执行将在运行相应代理镜像的 OpenShift Container Platform pod 下执行。

Jenkins 镜像还为 Kubernetes 插件提供附加代理镜像的自动发现和自动配置。使用 OpenShift Sync 插件，Jenkins 上的 Jenkins 镜像会在其正在运行的项目中搜索，或搜索在插件配置中特别列出的项目：

- 将标签 **role** 设置为 **jenkins-slave** 的镜像流。
- 将注解 **role** 设置为 **jenkins-slave** 的镜像流标签。
- 将标签 **role** 设置为 **jenkins-slave** 的 ConfigMaps。

当找到具有适当标签的镜像流或带有适当注解的镜像流标签时，它会生成对应的 Kubernetes 插件配置，以便您可以将 Jenkins 任务分配在运行镜像流提供的容器镜像的 pod 中运行。

镜像流或镜像流标签的名称和镜像引用映射到 Kubernetes 插件 pod 模板中的名称和镜像字段。您可以通过使用 **slave-label** 键在镜像流或镜像流标签对象上设置注解，来控制 Kubernetes 插件 pod 模板的标签字段。否则，名称将用作标签。

当找到具有适当标签的 ConfigMap 时，它会假定 ConfigMap 的键值数据有效负载中的任何值都包含与 Jenkins 和 Kubernetes 插件 pod 模板的配置格式一致的 XML。使用 ConfigMaps 而非镜像流或镜像流标签时，需要注意的一个关键区别在于您可以控制 Kubernetes 插件 pod 模板的所有字段。

以下是 ConfigMap 示例：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-slave
data:
  template1: |-
    <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
      <inheritFrom></inheritFrom>
      <name>template1</name>
      <instanceCap>2147483647</instanceCap>
      <idleMinutes>0</idleMinutes>
      <label>template1</label>
      <serviceAccount>jenkins</serviceAccount>
      <nodeSelector></nodeSelector>
      <volumes/>
      <containers>
        <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
          <name>jnlp</name>
          <image>openshift/jenkins-agent-maven-35-centos7:v3.10</image>
          <privileged>false</privileged>
          <alwaysPullImage>true</alwaysPullImage>
          <workingDir>/tmp</workingDir>
          <command></command>
          <args>${computer.jnlpmac} ${computer.name}</args>
          <ttyEnabled>false</ttyEnabled>
          <resourceRequestCpu></resourceRequestCpu>
          <resourceRequestMemory></resourceRequestMemory>
          <resourceLimitCpu></resourceLimitCpu>
          <resourceLimitMemory></resourceLimitMemory>
          <envVars/>
        </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
      </containers>
      <envVars/>
      <annotations/>
      <imagePullSecrets/>
      <nodeProperties/>
    </org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
```

启动后，OpenShift Sync 插件会监控 OpenShift Container Platform 的 API 服务器用于对 **ImageStreams**、**ImageStreamTags** 和 **ConfigMaps** 的更新，并调整 Kubernetes 插件的配置。

特别是将应用以下规则：

- 从 **ConfigMap**、**ImageStream** 或 **ImageStreamTag** 中移除标签或注解会导致从 Kubernetes 插件配置中删除任何现有的 **PodTemplate**。
- 如果删除了这些对象，相应配置也会从 Kubernetes 插件中删除。

- 以前，创建适当标记或注解的 **ConfigMap**、**ImageStream** 或 **ImageStreamTag** 对象，或者在初始创建后添加标签，都会导致在 Kubernetes-plugin 配置中创建 **PodTemplate**。
- 对于使用 **ConfigMap** 表单的 **PodTemplate**，对 **PodTemplate** 的 **ConfigMap** 数据的更改将应用到 Kubernetes 插件配置中的 **PodTemplate** 设置，，并将覆盖在 **ConfigMap** 更改之间临时通过 Jenkins UI 对 **PodTemplate** 所做的任何更改。

要将容器镜像用作 Jenkins 代理，该镜像必须运行 slave 代理，作为入口点。有关此方面的更多详情，请参阅官方 Jenkins 文档。

4.2.3.6.1. 权限注意事项

在以前的版本中，Pod 模板 XML 的 **<serviceAccount>** 元素是用于生成的 Pod 的 OpenShift Container Platform 服务账户。挂载到 Pod 的服务帐户凭证，以及与服务帐户关联的权限，控制允许从 Pod 针对 OpenShift Container Platform master 执行的操作。

考虑使用 OpenShift Container Platform Jenkins 镜像中运行的 Kubernetes 插件启动的用于 Pod 的服务帐户：

- 如果您使用 OpenShift Container Platform 提供的 Jenkins 示例模板，则 **jenkins** 服务帐户将由运行 Jenkins 的项目的 **edit** 角色定义，且 master Jenkins Pod 已挂载了该服务帐户。
- 注入 Jenkins 配置的两个默认 Maven 和 NodeJS Pod 模板也被设置为使用与 master 相同的服务帐户。
- 由于镜像流标签（tag）具有所需标签或注解，其服务帐户被 OpenShift 的服务帐户设置为 master 的服务帐户，因此 OpenShift Sync 插件会自动发现任何 Pod 模板。
- 对于其他方法，您可在 Jenkins 和 Kubernetes 插件中提供 Pod 模板定义，但必须明确指定要使用的服务帐户。
- 其它方法包括 Jenkins 控制台、由 Kubernetes 插件提供的 **podTemplate** 管道 DSL，或标记其数据为 Pod 模板的 XML 配置的 ConfigMap。
- 如果没有为服务帐户指定值，则将使用 **default** 服务帐户。
- 确保所使用的任何服务帐户均具有 OpenShift Container Platform 中定义的必要权限、角色等，以操作您选择从 Pod 中操作的任何项目。

4.2.4. 使用方法

4.2.4.1. 从模板创建 Jenkins 服务

模板 提供参数字段来定义所有带有预定义的默认值的环境变量（password）。OpenShift Container Platform 提供模板，以简化 Jenkins 服务的新建操作。在初始集群设置过程中，集群管理员应该在默认 **openshift** 项目中注册 Jenkins 模板。如需了解更多详细信息，请参阅 载入默认镜像流和模板。

部署配置和服务。

注意

当 Pod 移到另一节点，或当对部署配置的更新触发了重新部署时，Pod 可能会重启。

- **jenkins-persistent** 使用持久性卷存储。数据不会因 Pod 重启而丢失。

必须实例化模板才能使用 Jenkins:

4.2.4.2. 使用 Jenkins Kubernetes 插件

创建新 Jenkins 服务

在以下示例中，openshift-jee-sample BuildConfig 会导致 Jenkins maven 代理 Pod 动态置备。Pod 会克隆一些 Java 源，构建一个 WAR 文件，然后导致第二个 BuildConfig（openshift-jee-sample-docker）将新创建的 WAR 文件分层到一个容器镜像中。

这里 提供了实现类似目的的更完整的示例。

例 4.1. 使用 Jenkins Kubernetes 插件的 BuildConfig 示例

```
kind: List
apiVersion: v1
items:
- kind: ImageStream
  apiVersion: v1
  metadata:
    name: openshift-jee-sample
- kind: BuildConfig
  apiVersion: v1
  metadata:
    name: openshift-jee-sample-docker
  spec:
    strategy:
      type: Docker
    source:
      type: Docker
      dockerfile: |-
        FROM openshift/wildfly-101-centos7:latest
        COPY ROOT.war /wildfly/standalone/deployments/ROOT.war
        CMD $STI_SCRIPTS_PATH/run
      binary:
        asFile: ROOT.war
    output:
      to:
        kind: ImageStreamTag
        name: openshift-jee-sample:latest
- kind: BuildConfig
  apiVersion: v1
  metadata:
    name: openshift-jee-sample
  spec:
    strategy:
      type: JenkinsPipeline
      jenkinsPipelineStrategy:
        jenkinsfile: |-
          node("maven") {
            sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
            sh "mvn -B -Popenshift package"
            sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
          }
    triggers:
    - type: ConfigChange
```

-

它还可覆盖动态创建的 Jenkins 代理 Pod 的规范。以下是对上例的修改，可覆盖容器内存并指定环境变量：

例 4.2. 使用 Jenkins Kubernetes 插件的 BuildConfig 示例，指定内存限制和环境变量

```
kind: BuildConfig
apiVersion: v1
metadata:
  name: openshift-jee-sample
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        podTemplate(label: "mypod", 1
                cloud: "openshift", 2
                inheritFrom: "maven", 3
                containers: [
          containerTemplate(name: "jnlp", 4
                    image: "openshift/jenkins-agent-maven-35-centos7:v3.10", 5
                    resourceRequestMemory: "512Mi", 6
                    resourceLimitMemory: "512Mi", 7
                    envVars: [
            envVar(key: "CONTAINER_HEAP_PERCENT", value: "0.25") 8
          ])
        ]) {
          node("mypod") { 9
            sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
            sh "mvn -B -Popenshift package"
            sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
          }
        }
  triggers:
  - type: ConfigChange
```

1. 一个名为 "mypod" 的新 Pod 模板会自行定义。新 Pod 模板名称在节点片段中引用。
2. “cloud” 值必须设置为 “openshift”。
3. 新 Pod 模板可继承现有 Pod 模板的配置。在本例中，我们继承了 OpenShift Container Platform 预定义的"maven" Pod 模板。
4. 我们是在预先存在的容器中覆盖的值，因此我们必须根据名称指定它。OpenShift Container Platform 附带的所有 Jenkins 代理镜像均使用容器名称 “jnlp”。
5. 容器镜像必须重新指定。这是个已知问题。
6. 指定了 512Mi 的内存请求。
7. 指定了 512Mi 的内存限值。
8. CONTAINER_HEAP_PERCENT 环境变量，其值指定为 “0.25”。

9 节点片段引用上方定义的 Pod 模板的名称。

构建完成后会默认删除 pod。此行为可通过插件或在 Jenkinsfile 管道中修改 - 请参阅 代理 Pod 保留 以了解更多详细信息。

如需有关 Kubernetes 插件配置的更多信息,请参阅 Kubernetes 插件文档。

4.2.4.3. 内存要求

使用所提供的 Jenkins Ephemeral 或 Jenkins Persistent 模板部署时，默认内存限值为 512MiB。

如需了解有关调整 Jenkins 使用的 JVM 的后台信息，请参阅 在 OpenShift Container Platform 上调整 OpenJDK。

为提高内存效率，如果容器中运行的内存限值为 2GiB，则默认情况下 Jenkins 镜像会动态使用 32 位 JVM。**OPENSHIFT_JENKINS_JVM_ARCH** 环境变量可覆盖此行为。

默认情况下，Jenkins JVM 将容器内存限值的 50% 用于其堆。该值可通过 **CONTAINER_HEAP_PERCENT** 环境变量修改，还可设置上限或整个覆盖。如需了解更多详细信息，请参阅 环境变量。

注意默认情况下，Jenkins 容器中执行的所有进程（如 shell 脚本或从管道中运行的 **oc** 命令）不可能在不引发 OOM 终止的情况下使用剩余的 256MiB 内存。因此，我们强烈建议管道尽可能在代理容器中运行外部命令。

建议在由 Jenkins Kubernetes 插件创建的代理容器上指定内存请求和限值。作为 admin，可通过 Jenkins 配置基于每个代理镜像设置默认值。如上所述，内存请求和限制也可以基于每个容器覆盖。

在实例化 Jenkins Ephemeral 或 Jenkins Persistent 模板时，您可通过覆盖 **MEMORY_LIMIT** 参数来增加 Jenkins 的可用内存量。

4.2.5. Jenkins 插件

提供以下插件用于将 Jenkins 与 OpenShift Container Platform 集成。它们默认在 Jenkins 镜像中可用。

4.2.5.1. OpenShift Container Platform 客户端插件

OpenShift Container Platform 客户端插件旨在提供易读、简洁、全面且流畅的 Jenkins Pipeline 语法,以便与 OpenShift Container Platform 进行丰富的交互。该插件利用 **oc** 二进制文件，该二进制文件必须在执行脚本的节点上可用。

此插件被完全支持，并包含在 Jenkins 镜像中。它提供：

- Jenkins Pipelines 中使用 Fluent 风格的语法。
- 使用 **oc** 以外的任何选项。
- 与 Jenkins 凭证和集群集成。
- 继续支持经典 Jenkins 自由风格的任务。

如需更多信息，请参阅 OpenShift Pipeline Builds 指南 和插件的 README。

4.2.5.2. OpenShift Container Platform Pipeline 插件

OpenShift Container Platform Pipeline 插件是 Jenkins 和 OpenShift Container Platform 之前的一个集成，它的功能比 OpenShift Container Platform 客户端插件要少。它已被弃用，但仍可在 OpenShift Container Platform 版本中使用 v3.11。对于以后的 OpenShift Container Platform 版本，可以直接从 Jenkins Pipelines 中使用 **oc** 二进制文件，也可以使用 OpenShift Container Platform 客户端插件。

如需更多信息，请参阅插件的 README。

4.2.5.3. OpenShift Container Platform 同步插件

为便于 OpenShift Container Platform Pipeline 构建策略 在 Jenkins 和 OpenShift Container Platform 间集成，OpenShift Sync 插件 监控 OpenShift Container Platform 的 API 服务器，以获取对 **BuildConfigs** 和 **Builds** 的更新，以使用 Pipeline 策略并创建 Jenkins Pipeline 项目（在创建 **BuildConfig** 时）或者在生成的项目中启动作业（在 **Build** 启动时）。

如 配置 Jenkins Kubernetes 插件中 所述，此插件可根据 OpenShift Container Platform 中定义的 **ImageStream**、**ImageStreamTag** 或 **ConfigMap** 对象的具体情况，为 Kubernetes 插件创建 **PodTemplate** 配置。

此插件现在可以使用一个带有**credentials.sync.jenkins.openshift.io** 标签键，标签值为 **true** 的 **Secret** 对象，并构建 Jenkins 凭证，这些凭证放置在 Jenkins 凭证层次结构中的默认全局域中。凭证 ID 由在中定义 **Secret** 的命名空间组成、一个连字符（**-**），后跟 **Secret** 的名称。

与处理 **PodTemplates** 的 **ConfigMap** 类似，OpenShift Container Platform 中定义的 **Secret** 对象被视为主配置。OpenShift Container Platform 中对象的任何后续更新将应用于 Jenkins 凭证（覆盖期间对凭证所做的任何更改）。

移除了 **credential.sync.jenkins.openshift.io** 属性，将该属性设置为 **true** 以外的任何属性，或删除 OpenShift Container Platform 中的 **Secret**，会导致删除 Jenkins 中的相关凭证。

secret 的类型将映射到 jenkins 凭证类型，如下所示：

- Opaque 类型的 **Secret** 对象，插件会在 **data** 部分查找 **username** 和 **password**，并组成一个 Jenkins UsernamePasswordCredentials 凭证。请记住，在 OpenShift Container Platform 中，**password** 字段可以是实际密码，也可以是用户的唯一令牌。如果没有这些，它会查找 **ssh-privatekey** 字段并创建一个 Jenkins BasicSSHUserPrivateKey 凭证。
- 使用 **kubernetes.io/basic-auth** 类型 'Secret'objects 会创建一个 Jenkins UsernamePasswordCredentials 凭证。
- 使用 **kubernetes.io/ssh-auth** 类型 **Secret** 对象，该插件会创建一个 Jenkins BasicSSHUserPrivateKey 凭证。

4.2.5.4. Kubernetes 插件

Kubernetes 插件用于将 Jenkins 代理作为集群中的 pod 运行。使用 Jenkins Kubernetes 插件 中提供了 Kubernetes 插件的自动配置信息。

4.3. JENKINS 代理

4.3.1. 概述

OpenShift Container Platform 提供了三个适合用作 Jenkins 代理的镜像：***Base***、***Maven*** 和 ***Node.js*** 镜像。

第一个是适用于 Jenkins 代理的基础镜像：

- 它会拉取（pull）所需工具、无头 Java、Jenkins JNLP 客户端以及一些实用工具，其中包括 git、tar、zip 和 nss 等。
- 它将 JNLP 代理设立为入口点。
- 它包含 **oc** 客户端工具，用于从 Jenkins 任务调用命令行操作。
- 它为 CentOS 和 RHEL 镜像提供了 Dockerfile。

另外还提供了扩展基础镜像的两个镜像：

- Maven v3.5 镜像
- Node.js v10 镜像和 Node.js v12 镜像

Maven 和 Node.js Jenkins 代理镜像为 CentOS 和 RHEL 提供 Dockerfile，您可在构建新代理镜像时引用。另请注意 **contrib** 和 **contrib/bin** 子目录。这些子目录可用于为您的镜像插入配置文件和可执行脚本。

重要

为您要使用的 OpenShift Container Platform 版本使用并扩展适当的代理镜像版本。如果嵌入在代理镜像中的 **oc** 客户端版本与 OpenShift Container Platform 版本不兼容，则可能引发意外行为。如需更多信息，请参阅 版本策略。

4.3.2. 镜像

OpenShift Container Platform Jenkins 代理镜像有两种类型：

基于 RHEL 7 的镜像

RHEL 7 镜像可以通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/openshift3/jenkins-slave-base-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-slave-maven-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-slave-nodejs-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-agent-maven-35-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-agent-nodejs-10-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-agent-nodejs-12-rhel7
```

基于 CentOS 7 的镜像

这些镜像在 Docker Hub 上可用：

```
$ docker pull openshift/jenkins-slave-base-centos7
$ docker pull openshift/jenkins-slave-maven-centos7
$ docker pull openshift/jenkins-slave-nodejs-centos7
$ docker pull openshift/jenkins-agent-maven-35-centos7
$ docker pull openshift/jenkins-agent-nodejs-10-centos7
$ docker pull openshift/jenkins-agent-nodejs-12-centos7
```

要使用这些镜像，您可直接从这些 registry 访问镜像或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。

4.3.3. 配置和自定义

4.3.3.1. 环境变量

每个 Jenkins 代理容器均可通过以下环境变量进行配置：

- **OPENSHIFT_JENKINS_JVM_ARCH**
  设置为 **x86_64** 或 **i386**，以覆盖用来托管 Jenkins 代理的 JVM。为提高内存效率，如果容器中运行的内存限值为 2GiB，则 Jenkins 代理镜像默认会动态使用 32 位 JVM。

- **JAVA_MAX_HEAP_PARAM**
  CONTAINER_HEAP_PERCENT（默认值： **0.1**、e.e. 10%）
  JNLP_MAX_HEAP_UPPER_BOUND_MB
  这些值控制 Jenkins 代理 JVM 的最大堆大小。如果设置了 **JAVA_MAX_HEAP_PARAM**（示例设置： **-Xmx512m**），则优先使用其值。否则，最大堆大小将动态计算为容器内存限制的 **CONTAINER_HEAP_PERCENT**%（示例设置： **0.5**、例如 50%），可选上限为 **JNLP_MAX_HEAP_UPPER_BOUND_MB** MiB（示例设置： **512**）。

  默认情况下，Jenkins 代理 JVM 的最大堆大小设置为容器内存限值的 50%，且无上限。

- **JAVA_INITIAL_HEAP_PARAM**
  CONTAINER_INITIAL_PERCENT
  这些值控制 Jenkins 代理 JVM 的初始堆大小。如果设置了 **JAVA_INITIAL_HEAP_PARAM**（示例设置： **-Xms32m**），则优先使用其值。否则，初始堆大小可能会动态计算为动态计算的最大堆大小的 **CONTAINER_INITIAL_PERCENT**%（示例： **0.1**，即 10%）。

  默认情况下，初始堆大小留给 JVM。

- **CONTAINER_CORE_LIMIT**
  如果设置，请将用于调整内部 JVM 线程数的内核数指定为整数。示例设置：**2**。

- **JAVA_TOOL_OPTIONS** (default: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**)
  指定此容器中运行的所有 JVM 需要注意的选项。不建议覆盖此选项。

- **JAVA_GC_OPTS** (default: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaiveSizePolicyWeight=90**)
  指定 Jenkins 代理 JVM 垃圾回收参数。不建议覆盖此选项。

- **JNLP_JAVA_OVERRIDES**
  指定 Jenkins 代理 JVM 的额外选项。这些选项附加到所有其他选项中，包括上面的 Java 选项，必要时可用于覆盖其中任何一个选项。用空格分开各个附加选项；如有任意选项包含空格字符，请使用反斜杠转义。示例设置：**-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value**。

4.3.4. 使用方法

4.3.4.1. 内存要求

所有 Jenkins 代理均使用 JVM 来托管 Jenkins JNLP 代理和运行任何 Java 应用程序，如 **javac**、Maven 或 Gradle。如需了解有关调整 Jenkins 代理使用的 JVM 的背景资料，请参阅在 OpenShift Container Platform 上调整 OpenJDK。

为提高内存效率，如果容器中运行的内存限值为 2GiB，则默认情况下 Jenkins 镜像会动态使用 32 位 JVM。**OPENSHIFT_JENKINS_JVM_ARCH** 环境变量可覆盖此行为。JVM 选择默认适用于 Jenkins JNLP 代理以及代理容器内的任何其他 Java 进程。

默认情况下，Jenkins JNLP 代理 JVM 会将容器内存限值的 50% 用于其堆。该值可通过 **CONTAINER_HEAP_PERCENT** 环境变量修改，还可设置上限或整个覆盖。如需了解更多详细信息，请参阅环境变量。

在默认情况下，Jenkins 代理容器中执行的任何/所有其他进程（如 shell 脚本或从管道运行的 **oc** 命令）在不引发 OOM 终止的情况下，可能无法使用剩余的 50% 内存限值。

默认情况下，Jenkins 代理容器中运行的每个进一步的 JVM 进程最多可为其堆使用 25% 的容器内存限值。可能需要对许多构建工作负载进行微调。如需更多信息，请参阅 在 OpenShift Container Platform 上调整 OpenJDK。

如需有关指定 Jenkins 代理容器的内存请求和限制的信息，请参阅 Jenkins 文档。

4.3.4.1.1. gradle 构建

在 OpenShift 上的 Jenkins 代理中托管 Gradle 构建会带来额外的复杂情况，特别是除了 Jenkins JNLP 代理和 Gradle JVM 外，Gradle 还会生成第三个 JVM 来运行测试（如果指定了这些测试）。

如需了解在 OpenShift Container Platform 上调整 JVM 的后台信息，请参阅 OpenShift Container Platform 上调整 OpenJDK。

建议进行以下设置，以便在 OpenShift 上内存受限的 Jenkins 代理中运行 Gradle 构建。可根据需要放松设置。

- 通过将 **org.gradle.daemon=false** 添加到 gradle.properties 文件中来确保禁用长期 Gradle 守护进程。
- 通过确保 gradle.properties 文件中未设置 **org.gradle.parallel=true** 且 **--parallel** 未设置为命令行参数来禁用并行构建执行。
- 要防止 Java 编译超出进程范围，在 build.gradle 文件中设置 **java { options.fork = false }**。
- 通过确保在 build.gradle 文件中设置 **test { maxParallelForks = 1 }** 来禁用多个附加测试进程。
- 根据 OpenShift Container Platform 上调整 OpenJDK 的内容， 使用 GRADLE_OPTS、JAVA_OPTS 或 JAVA_TOOL_OPTIONS 环境变量覆盖 gradle JVM 内存参数。
- 通过在 build.gradle 中定义 maxHeapSize 和 jvmArgs 设置，或通过 **-Dorg.gradle.jvmargs** 命令行参数来为任何 Gradle 测试 JVM 设置最大堆大小和 JVM 参数。

4.3.5. 代理 Pod 保留

构建完成或停止后，默认删除 Jenkins 代理 pod（也称为 slave pod）。该行为可通过设置 Kubernetes 插件的*Pod 保留*设置来修改。Pod 保留可针对所有 Jenkins 构建设置，并覆盖每个 pod 模板。支持以下行为：

- *Always* 保留构建 pod，不受构建结果的限制。
- *Default* 使用插件值（仅限 pod 模板）。
- *Never* 始终删除 pod。
- *On Failure* 如果构建过程中失败，则保留 pod。

您可覆盖管道 Jenkinsfile 中的 pod 保留：

```
podTemplate(label: "mypod",
  cloud: "openshift",
  inheritFrom: "maven",
  podRetention: onFailure(), 1
  containers: [
    ...
  ]) {
  node("mypod") {
    ...
  }
}
```

1 **podRetention** 允许的值为 **never()**、**onFailure()**、**always()** 和 **default()**。

警告

保留的 Pod 可能会根据资源配额继续运行和计数。

4.4. 其他容器镜像

如果要使用 Red Hat Container Catalog 中未找到的容器镜像，您可以在 OpenShift Container Platform 实例中使用其他任意容器镜像，例如在 Docker Hub 上找到的容器镜像。

有关使用任意分配的用户 ID 运行容器的 OpenShift Container Platform 特定准则，请参阅 创建镜像指南中的支持任意用户 ID。

重要

如需支持性的详细信息，请参阅 OpenShift Container Platform 支持策略中定义的产品支持范围。

另请参阅 系统和环境要求中的安全性警告。