



# builds for Red Hat OpenShift 1.0

使用构建

使用构建



使用构建

## 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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js<sup>®</sup> 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<sup>®</sup> 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

本文档提供了使用构建所需的流程示例。

---

## 目录

<b>第1章 运行构建</b> .....	<b>3</b>
1.1. 创建 BUILDDAH 构建	3
1.2. 创建 SOURCE-TO-IMAGE 构建	6
1.3. 查看日志	9
1.4. 删除资源	10
1.5. 其他资源	10



## 第 1 章 运行构建

安装构建后，您可以创建一个 **buildah** 或 **source-to-image** 构建以供使用。您还可以删除构建不需要的自定义资源。

### 1.1. 创建 BUILDDAH 构建

您可以创建一个 **buildah** 构建并将创建的镜像推送到目标 registry。

#### 先决条件

- 您已在 OpenShift Container Platform 集群中安装了 Red Hat OpenShift Operator 的 Builds。
- 您已创建了 **ShipwrightBuild** 资源。
- 已安装 **oc** CLI。
- 可选：已安装 **shp** CLI。

#### 流程

1. 使用其中一个 CLI 创建 **Build** 资源并将其应用到 OpenShift Container Platform 集群：

#### 示例：使用 oc CLI

```
$ oc apply -f - <<EOF
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-golang-build
spec:
  source: ①
    git:
      url: https://github.com/shipwright-io/sample-go
      contextDir: docker-build
  strategy: ②
    name: buildah
    kind: ClusterBuildStrategy
  paramValues: ③
    - name: dockerfile
      value: Dockerfile
  output: ④
    image: image-registry.openshift-image-registry.svc:5000/buildah-example/sample-go-app
EOF
```

- ① 放置源代码的位置。
- ② 用于构建容器的构建策略。
- ③ 构建策略中定义的参数。要设置 **dockerfile** 策略参数的值，请指定构建输出镜像所需的 Dockerfile 位置。
- ④ 推送构建镜像的位置。在本例中，构建的镜像被推送到 OpenShift Container Platform 集群内部 registry。**buildah-example** 是当前项目的名称。确保存在指定的项目以允许推送镜像。

例。

### 示例：使用 shp CLI

```
$ shp build create buildah-golang-build \
--source-url="https://github.com/shipwright-io/sample-go" --source-context-dir="docker-build" \
1
--strategy-name="buildah" 2
--dockerfile="Dockerfile" 3
--output-image="image-registry.openshift-image-registry.svc:5000/buildah-example/go-app"
4
```

- 1 放置源代码的位置。
- 2 用于构建容器的构建策略。
- 3 构建策略中定义的参数。要设置 **dockerfile** 策略参数的值，请指定构建输出镜像所需的 Dockerfile 位置。
- 4 推送构建镜像的位置。在本例中，构建的镜像被推送到 OpenShift Container Platform 集群内部 registry。**buildah-example** 是当前项目的名称。确保存在指定的项目以允许推送镜像。

2. 使用其中一个 CLI 检查 **Build** 资源是否已创建：

### 示例：使用 oc CLI

```
$ oc get builds.shipwright.io buildah-golang-build
```

### 示例：使用 shp CLI

```
$ shp build list
```

3. 使用其中一个 CLI 创建 **BuildRun** 资源并将其应用到 OpenShift Container Platform 集群：

### 示例：使用 oc CLI

```
$ oc apply -f - <<EOF
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: buildah-golang-buildrun
spec:
  build:
    name: buildah-golang-build 1
EOF
```

- 1 **spec.build.name** 字段表示要运行的相应构建，该构建预期在同一命名空间中可用。

### 示例：使用 shp CLI

■

```
$ shp build run buildah-golang-build --follow 1
```

- 1 可选：通过使用 **--follow** 标志，您可以在输出结果中查看构建日志。

4. 使用其中一个 CLI 检查 **BuildRun** 资源是否已创建：

示例：使用 **oc** CLI

```
$ oc get buildrun buildah-golang-buildrun
```

示例：使用 **shp** CLI

```
$ shp buildrun list
```

**BuildRun** 资源会创建一个 **TaskRun** 资源，然后创建 pod 来执行构建策略步骤。

## 验证

1. 所有容器完成其任务后，验证以下内容：

- 检查 pod 是否将 **STATUS** 字段显示为 **Completed**：

```
$ oc get pods -w
```

输出示例

NAME	READY	STATUS	RESTARTS	AGE
buildah-golang-buildrun-dtrg2-pod	2/2	Running	0	4s
buildah-golang-buildrun-dtrg2-pod	1/2	NotReady	0	7s
buildah-golang-buildrun-dtrg2-pod	0/2	Completed	0	55s

- 检查对应的 **TaskRun** 资源是否将 **SUCCEEDED** 字段显示为 **True**：

```
$ oc get tr
```

输出示例

NAME	SUCCEEDED	REASON	STARTTIME	COMPLETIONTIME
buildah-golang-buildrun-dtrg2	True	Succeeded	11m	8m51s

- 检查对应的 **BuildRun** 资源是否将 **SUCCEEDED** 字段显示为 **True**：

```
$ oc get br
```

输出示例

NAME	SUCCEEDED	REASON	STARTTIME	COMPLETIONTIME
buildah-golang-buildrun	True	Succeeded	13m	11m

在验证过程中，如果构建运行失败，您可以检查 **BuildRun** 资源中的 **status.failureDetails** 字段，以识别 pod 或容器中发生故障的确切点。



### 注意

pod 可能会切换到 **NotReady** 状态，因为其中一个容器已经完成其任务。这是预期的行为。

- 验证镜像是否已推送到 **build.spec.output.image** 字段中指定的 registry。您可以运行以下命令来从可以访问内部 registry 的节点运行以下命令来拉取镜像：

```
$ podman pull image-registry.openshift-image-registry.svc:5000/<project>/<image> 1
```

- 1 创建 **Build** 资源时使用的项目名称和镜像名称。例如，您可以使用 **buildah-example** 作为项目名称，**example-go-app** 作为镜像名称。

## 1.2. 创建 SOURCE-TO-IMAGE 构建

您可以创建一个 **source-to-image** 构建，并将创建的镜像推送到自定义 Quay 存储库。

### 先决条件

- 您已在 OpenShift Container Platform 集群中安装了 Red Hat OpenShift Operator 的 Builds。
- 您已创建了 **ShipwrightBuild** 资源。
- 已安装 **oc** CLI。
- 可选：已安装 **shp** CLI。

### 流程

- 使用其中一个 CLI 创建 **Build** 资源并将其应用到 OpenShift Container Platform 集群：

#### 示例：使用 oc CLI

```
$ oc apply -f - <<EOF
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: s2i-nodejs-build
spec:
  source: 1
  git:
    url: https://github.com/shipwright-io/sample-nodejs
    contextDir: source-build/
  strategy: 2
  name: source-to-image
  kind: ClusterBuildStrategy
  paramValues: 3
  - name: builder-image
    value: quay.io/centos7/nodejs-12-centos7
```

```
output:
  image: quay.io/<repo>/s2i-nodejs-example ④
  pushSecret: registry-credential ⑤
EOF
```

- ① 放置源代码的位置。
- ② 用于构建容器的构建策略。
- ③ 构建策略中定义的参数。要设置 **builder-image** 策略参数的值，请指定构建输出镜像所需的构建器镜像位置。
- ④ 推送构建镜像的位置。您可以将构建的镜像推送到自定义 Quay.io 存储库。将 **repo** 替换为有效的 Quay.io 机构或 Quay 用户名。
- ⑤ 存储用于推送容器镜像的凭据的机密名称。要为身份验证生成 **docker-registry** 类型的 secret，请参阅 "Authentication to container registries"。

### 示例：使用 shp CLI

```
$ shp build create s2i-nodejs-build \
--source-url="https://github.com/shipwright-io/sample-nodejs" --source-context-dir="source-
build" ①
--strategy-name="source-to-image" ②
--builder-image="quay.io/centos7/nodejs-12-centos7" ③
--output-image="quay.io/<repo>/s2i-nodejs-example" ④
--output-credentials-secret="registry-credential" ⑤
```

- ① 放置源代码的位置。
- ② 用于构建容器的构建策略。
- ③ 构建策略中定义的参数。要设置 **builder-image** 策略参数的值，请指定构建输出镜像所需的构建器镜像位置。
- ④ 推送构建镜像的位置。您可以将构建的镜像推送到自定义 Quay.io 存储库。将 **repo** 替换为有效的 Quay.io 机构或 Quay 用户名。
- ⑤ 存储用于推送容器镜像的凭据的机密名称。要为身份验证生成 **docker-registry** 类型的 secret，请参阅 "Authentication to container registries"。

2. 使用其中一个 CLI 检查 **Build** 资源是否已创建：

### 示例：使用 oc CLI

```
$ oc get builds.shipwright.io s2i-nodejs-build
```

### 示例：使用 shp CLI

```
$ shp build list
```

3. 使用其中一个 CLI 创建 **BuildRun** 资源并将其应用到 OpenShift Container Platform 集群：

**示例：使用 oc CLI**

```
$ oc apply -f - <<EOF
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: s2i-nodejs-buildrun
spec:
  build:
    name: s2i-nodejs-build ❶
EOF
```

- ❶ **spec.build.name** 字段表示要运行的相应构建，该构建预期在同一命名空间中可用。

**示例：使用 shp CLI**

```
$ shp build run s2i-nodejs-build --follow ❶
```

- ❶ 可选：通过使用 **--follow** 标志，您可以在输出结果中查看构建日志。

4. 使用其中一个 CLI 检查 **BuildRun** 资源是否已创建：

**示例：使用 oc CLI**

```
$ oc get buildrun s2i-nodejs-buildrun
```

**示例：使用 shp CLI**

```
$ shp buildrun list
```

**BuildRun** 资源会创建一个 **TaskRun** 资源，然后创建 pod 来执行构建策略步骤。

**验证**

1. 所有容器完成其任务后，验证以下内容：

- 检查 pod 是否将 **STATUS** 字段显示为 **Completed**：

```
$ oc get pods -w
```

**输出示例**

```
NAME                                READY STATUS  RESTARTS AGE
s2i-nodejs-buildrun-phxxm-pod      2/2   Running  0      10s
s2i-nodejs-buildrun-phxxm-pod      1/2   NotReady  0      14s
s2i-nodejs-buildrun-phxxm-pod      0/2   Completed 0       2m
```

- 检查对应的 **TaskRun** 资源是否将 **SUCCEDED** 字段显示为 **True**：

```
$ oc get tr
```

## 输出示例

```
NAME                SUCCEEDED REASON  STARTTIME  COMPLETIONTIME
s2i-nodejs-buildrun True      Succeeded  2m39s      13s
```

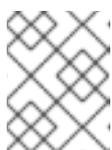
- 检查对应的 **BuildRun** 资源是否将 **SUCCEEDED** 字段显示为 **True** :

```
$ oc get br
```

## 输出示例

```
NAME                SUCCEEDED REASON  STARTTIME  COMPLETIONTIME
s2i-nodejs-buildrun True      Succeeded  2m41s      15s
```

在验证过程中，如果构建运行失败，您可以检查 **BuildRun** 资源中的 **status.failureDetails** 字段，以识别 pod 或容器中发生故障的确切点。



### 注意

pod 可能会切换到 **NotReady** 状态，因为其中一个容器已经完成其任务。这是预期的行为。

2. 验证镜像是否已推送到 **build.spec.output.image** 字段中指定的 registry。您可以在登录到 registry 后运行以下命令来尝试拉取镜像：

```
$ podman pull quay.io/<repo>/<image> 1
```

- 1** 创建 **Build** 资源时使用的存储库名称和镜像名称。例如，您可以使用 **s2i-nodejs-example** 作为镜像名称。

## 其他资源

- [容器 registry 身份验证](#)

## 1.3. 查看日志

您可以查看构建运行的日志，以识别任何运行时错误并解决它们。

### 先决条件

- 已安装 **oc** CLI。
- 可选：已安装 **shp** CLI。

### 流程

- 使用其中一个 CLI 查看构建运行的日志：

#### 使用 **oc** CLI

```
$ oc logs <buildrun_resource_name>
```

-

### 使用 shp CLI

```
$ shp buildrun logs <buildrun_resource_name>
```

## 1.4. 删除资源

如果项目中不需要 **Build**、**BuildRun** 或 **BuildStrategy** 资源，您可以删除 Build、BuildRun 或 BuildStrategy 资源。

### 先决条件

- 已安装 **oc** CLI。
- 可选：已安装 **shp** CLI。

### 流程

- 使用其中一个 CLI 删除 **构建** 资源：

#### 使用 oc CLI

```
$ oc delete builds.shipwright.io <build_resource_name>
```

#### 使用 shp CLI

```
$ shp build delete <build_resource_name>
```

- 使用其中一个 CLI 删除 **BuildRun** 资源：

#### 使用 oc CLI

```
$ oc delete buildrun <buildrun_resource_name>
```

#### 使用 shp CLI

```
$ shp buildrun delete <buildrun_resource_name>
```

- 运行以下命令来删除 **BuildStrategy** 资源：

#### 使用 oc CLI

```
$ oc delete buildstrategies <buildstrategy_resource_name>
```

## 1.5. 其他资源

- [容器 registry 身份验证](#)
- [使用 Web 控制台创建 ShipwrightBuild 资源](#)

