builds for Red Hat OpenShift 1.0

配置

配置构建

Last Updated: 2024-02-15

builds for Red Hat OpenShift 1.0 配置

配置构建

法律通告

摘要

本文档提供有关配置构建的信息。

目录

第 1 章 配置构建

Build 自定义资源(CR)可帮助您定义源、构建策略、参数值、输出、保留参数和卷来配置构建。**构建** 资源可在命名空间内使用。

要配置构建，请创建一个 **Build** 资源 YAML 文件并将其应用到 OpenShift Container Platform 集群。

1.1. 构建中的可配置字段

您可以在 **Build** 自定义资源(CR)中使用以下字段：

表 1.1. Build CR 中的字段

字段	存在	描述
apiVersion	必填	指定资源的 API 版本，例如 **shipwright.io/v1beta1**。
kind	必填	指定资源的类型，如 **Build**。
metadata	必填	表示标识自定义资源定义实例的元数据，如 **Build** 资源的名称。
spec.source	必填	表示源代码的位置，如 Git 存储库或源捆绑包镜像。
spec.strategy	必填	表示用于 **Build** 资源的策略的名称和类型。
spec.output	必填	表示要推送生成的镜像的位置。
spec.output.pushSecret	必填	表示现有的 secret 以获取容器 registry 的访问。
spec.paramValues	选填	表示 name-value 列表，用于指定构建策略中定义的参数的值。
spec.timeout	选填	定义自定义超时。默认值为 10 分钟。您可以在 **BuildRun** 资源中覆盖此字段值。
spec.output.annotations	选填	表示可用于注解输出镜像的键值对列表。
spec.output.labels	选填	表示可用于标记输出镜像的键值对列表。
spec.env	选填	定义您可以传递给构建容器的额外环境变量。可用的变量取决于您的构建策略使用的工具。
spec.retention.ttlAfterFailed	选填	指定失败的构建运行可以存在的持续时间。
spec.retention.ttlAfterSucceeded	选填	指定成功构建运行可以存在的持续时间。

字段	存在	描述
spec.retention.failedLimit	选填	指定可以存在的失败构建运行数量。
spec.retention.succeededLimit	选填	指定可存在的构建运行数量。

1.2. 源定义

您可以通过设置以下字段的值，在 **Build** 自定义资源(CR)中为构建配置源详情：

- **source.git.url** ：定义 Git 存储库中可用的镜像的源位置。
- **source.git.cloneSecret** ：引用命名空间中包含私有 Git 存储库的 SSH 私钥的 secret。
- **source.git.revision** ：定义从源 Git 存储库选择的特定修订版本。例如，提交、标签或分支名称。此字段默认为 Git 存储库默认分支。
- **source.contextDir** ：指定根文件夹中不存在源代码的存储库的上下文路径。

构建控制器不会自动验证您为拉取镜像指定的 Git 存储库是否存在。如果您需要验证，将 **build.shipwright.io/verify.repository** 注解的值设置为 **true**，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-golang-build
  annotations:
    build.shipwright.io/verify.repository: "true"
spec:
  source:
    git:
      url: https://github.com/shipwright-io/sample-go
    contextDir: docker-build
```

构建控制器在以下情况下验证 Git 存储库是否存在：

- 当您将端点 URL 与 HTTP 或 HTTPS 协议一起使用时。
- 当您定义了 SSH 协议时，如 **git@**，而不是引用的 secret，如 **source.git.cloneSecret**。

以下示例演示了如何使用不同的源输入集配置构建。

示例：使用凭证配置构建

您可以通过指定凭证来使用源配置构建，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-build
spec:
  source:
```

```
    git:
      url: https://github.com/sclorg/nodejs-ex
      cloneSecret: source-repository-credentials
```

示例：使用上下文路径配置构建

您可以使用指定 Git 存储库中的上下文路径的源配置构建，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-custom-context-dockerfile
spec:
  source:
    git:
      url: https://github.com/userjohn/npm-simple
    contextDir: docker-build
```

示例：使用标签配置构建

您可以使用为 Git 存储库指定标签 **v.0.1.0** 的源配置构建，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-golang-build
spec:
  source:
    git:
      url: https://github.com/shipwright-io/sample-go
      revision: v0.1.0
```

示例：使用环境变量配置构建

您还可以配置指定环境变量的构建，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-golang-build
spec:
  source:
    git:
      url: https://github.com/shipwright-io/sample-go
    contextDir: docker-build
  env:
    - name: <example_var_1>
      value: "<example_value_1>"
    - name: <example_var_2>
      value: "<example_value_2>"
```

1.3. 策略定义

您可以在 **Build** CR 中为构建配置策略。以下构建策略可供使用：

- **buildah**
- **Source-to-image**

要配置构建策略，请在 **Build** CR 中定义 **spec.strategy.name** 和 **spec.strategy.kind** 字段，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-build
spec:
  strategy:
    name: buildah
    kind: ClusterBuildStrategy
```

1.4. 构建的参数值定义

您可以在 **Build** CR 中为构建策略参数指定值。通过指定参数值，您可以控制构建策略的工作方式。您还可以覆盖 **BuildRun** 资源中的值。

对于所有参数，您必须直接指定值，也可以使用配置映射或 secret 中的引用键指定值。

注意

在构建策略步骤中使用该参数限制了配置映射和 secret 的使用。只有命令、参数或环境变量中使用了参数时，才能使用配置映射和 secret。

在 **Build** CR 中使用 **paramValues** 字段时，请避免以下情况：

- 指定与 **BuildStrategy** CR 中定义的 **spec.parameters** 之一不匹配的 **spec.paramValues** 名称。
- 指定与 Shipwright 保留参数冲突的 **spec.paramValues** 名称。这些参数包括 **BUILDER_IMAGE**、**CONTEXT_DIR** 以及从 **shp-** 开始的任何名称。

另外，在 **Build** CR 中定义 **paramValues** 字段前，请确保了解您的策略的内容。

1.4.1. 定义参数值的配置示例

以下示例演示了如何在构建策略中定义参数，并使用 **Build** CR 为这些参数分配值。您还可以为 **Build** CR 中的 type **数组** 的参数分配一个值。

示例：在 ClusterBuildStrategy CR 中定义参数

以下示例显示了定义几个参数的 **ClusterBuildStrategy** CR：

```
apiVersion: shipwright.io/v1beta1
kind: ClusterBuildStrategy
metadata:
  name: buildah
spec:
  parameters:
    - name: build-args
      description: "The values for the args in the Dockerfile. Values must be in the format
```

```
KEY=VALUE."
      type: array
      defaults: []
    # ...
    - name: storage-driver
      description: "The storage driver to use, such as 'overlay' or 'vfs'."
      type: string
      default: "vfs"
# ...
steps:
# ...
```

示例：将值分配给 Build CR 中的参数

以上 **ClusterBuildStrategy** CR 定义了一个 **storage-driver** 参数，您可以在 **Build** CR 中指定 **storage-driver** 参数的值，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: <your_build>
  namespace: <your_namespace>
spec:
  paramValues:
  - name: storage-driver
    value: "overlay"
  strategy:
    name: buildah
    kind: ClusterBuildStrategy
  output:
  # ...
```

示例：创建 ConfigMap CR 以集中控制参数

如果要将 **storage-driver** 参数用于多个构建并集中控制其使用，您可以创建一个 **ConfigMap** CR，如下例所示：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: buildah-configuration
  namespace: <your_namespace>
data:
  storage-driver: overlay
```

您可以使用创建的 **ConfigMap** CR 作为 **Build** CR 中的参数值，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: <your_build>
  namespace: <your_namespace>
spec:
  paramValues:
  - name: storage-driver
```

```
    configMapValue:
      name: buildah-configuration
      key: storage-driver
  strategy:
    name: buildah
    kind: ClusterBuildStrategy
  output:
  # ...
```

示例：将值分配给 Build CR 中的 type 数组 的参数

您可以为类型 **数组** 的参数分配值。如果使用 **buildah** 策略，您可以定义一个 **registry-search** 参数来在特定 registry 中搜索镜像。以下示例演示了如何为 **registry-search** 数组参数分配值：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: <your_build>
  namespace: <your_namespace>
spec:
  paramValues:
  - name: storage-driver
    configMapValue:
      name: buildah-configuration
      key: storage-driver
  - name: registries-search
    values:
    - value: registry.redhat.io
  strategy:
    name: buildah
    kind: ClusterBuildStrategy
  output:
  # ...
```

示例：在构建 CR 中引用 secret

您可以引用 **registry-block** 数组参数的 secret，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: <your_build>
  namespace: <your_namespace>
spec:
  paramValues:
  - name: storage-driver
    configMapValue:
      name: buildah-configuration
      key: storage-driver
  - name: registries-block
    values:
    - secretValue: (1)
        name: registry-configuration
        key: reg-blocked
  strategy:
```

```
    name: buildah
    kind: ClusterBuildStrategy
  output:
  # ...
```

1 该值引用一个 secret。

1.5. 构建器或 DOCKER 文件定义

在 **Build** CR 中，您可以使用 **spec.paramValues** 字段指定包含构建输出镜像的工具的镜像。以下示例在 **Build** CR 中指定 **Dockerfile** 镜像：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-golang-build
spec:
  source:
    git:
      url: https://github.com/shipwright-io/sample-go
    contextDir: docker-build
  strategy:
    name: buildah
    kind: ClusterBuildStrategy
  paramValues:
  - name: dockerfile
    value: Dockerfile
```

您还可以在 **Build** CR 中使用 **构建器镜像** 作为 **source-to-image** 构建策略的一部分，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: s2i-nodejs-build
spec:
  source:
    git:
      url: https://github.com/shipwright-io/sample-nodejs
    contextDir: source-build/
  strategy:
    name: source-to-image
    kind: ClusterBuildStrategy
  paramValues:
  - name: builder-image
    value: docker.io/centos/nodejs-10-centos7
```

1.6. 输出定义

在 **Build** CR 中，您可以指定一个输出位置来推送镜像。当使用外部私有 registry 作为输出位置时，您必须指定一个 secret 以访问镜像。您还可以为输出镜像指定注解和标签。

注意

当您指定注解或标签时，输出镜像会被推送两次。第一个推送来自构建策略，第二个推送会更改镜像配置以添加注解和标签。

以下示例定义了推送镜像的公共 registry：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: s2i-nodejs-build
spec:
  source:
    git:
      url: https://github.com/shipwright-io/sample-nodejs
    contextDir: source-build/
  strategy:
    name: source-to-image
    kind: ClusterBuildStrategy
  paramValues:
  - name: builder-image
    value: docker.io/centos/nodejs-10-centos7
  output:
    image: image-registry.openshift-image-registry.svc:5000/build-examples/nodejs-ex
```

以下示例定义了推送镜像的私有 registry：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: s2i-nodejs-build
spec:
  source:
    git:
      url: https://github.com/shipwright-io/sample-nodejs
    contextDir: source-build/
  strategy:
    name: source-to-image
    kind: ClusterBuildStrategy
  paramValues:
  - name: builder-image
    value: docker.io/centos/nodejs-10-centos7
  output:
    image: us.icr.io/source-to-image-build/nodejs-ex
    pushSecret: icr-knbuild
```

以下示例定义了镜像的注解和标签：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: s2i-nodejs-build
spec:
  source:
    git:
```

```
    url: https://github.com/shipwright-io/sample-nodejs
  contextDir: source-build/
 strategy:
  name: source-to-image
  kind: ClusterBuildStrategy
 paramValues:
 - name: builder-image
  value: docker.io/centos/nodejs-10-centos7
 output:
  image: us.icr.io/source-to-image-build/nodejs-ex
  pushSecret: icr-knbuild
  annotations:
    "org.opencontainers.image.source": "https://github.com/org/repo"
    "org.opencontainers.image.url": "https://my-company.com/images"
  labels:
    "maintainer": "team@my-company.com"
    "description": "This is my cool image"
```

1.7. 构建的保留参数定义

您可以为以下目的定义保留参数：

- 指定已完成的构建运行可以存在的时长
- 指定构建可以存在的成功或失败的构建数量

保留参数提供了一种自动清理 **BuildRun** 实例或资源的方法。您可以在 **Build** CR 中设置以下保留参数的值：

- **retention.succeededLimit** ：定义构建可以存在的成功构建运行的数量。
- **retention.failedLimit** ：定义构建可以存在的失败构建数量。
- **retention.ttlAfterFailed** ：指定构建运行失败的持续时间。
- **retention.ttlAfterSucceeded**: 指定成功构建运行可以存在的持续时间。

以下示例显示了在 **Build** CR 中使用保留参数：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: build-retention-ttl
spec:
  source:
    git:
      url: "https://github.com/shipwright-io/sample-go"
    contextDir: docker-build
  strategy:
    kind: ClusterBuildStrategy
    name: buildah
  output:
  # ...
  retention:
    ttlAfterFailed: 30m
    ttlAfterSucceeded: 1h
```

```
    failedLimit: 10
    succeededLimit: 20
  # ...
```

注意

当您更改 **retention.failedLimit** 和 **retention.succeededLimit** 参数的值时，会在构建中应用这些更改时强制实施新的限制。但是，当您更改 **retention.ttlAfterFailed** 和 **retention.ttlAfterSucceeded** 参数的值时，新的保留持续时间只在新构建运行时强制使用。旧构建运行遵循旧的保留持续时间。如果您在 **BuildRun** 和 Build CR 中定义了保留持续时间，则 **Build Run** CR 中定义的保留持续时间会获得优先级。

1.8. 构建的卷定义

您可以在 **Build** CR 中定义卷。定义的卷覆盖 **BuildStrategy** 资源中指定的卷。如果卷没有被覆盖，则构建运行会失败。

以下示例显示了在 **Build** CR 中使用 **volumes** 字段：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: <build_name>
spec:
  source:
    git:
      url: https://github.com/example/url
  strategy:
    name: buildah
    kind: ClusterBuildStrategy
  paramValues:
  - name: dockerfile
    value: Dockerfile
  output:
    image: registry/namespace/image:latest
  volumes:
    - name: <your_volume_name>
      configMap:
        name: <your_configmap_name>
```

第 2 章 配置构建策略

BuildStrategy 或 **ClusterBuildStrategy** 自定义资源(CR)可帮助您定义策略参数、系统参数、步骤资源定义、注解和卷来配置构建策略。**BuildStrategy** 资源可用于命名空间，而 **ClusterBuildStrategy** 资源可用于整个集群。

要配置构建策略，请创建一个 **BuildStrategy** 或 **ClusterBuildStrategy** 资源 YAML 文件，并将其应用到 OpenShift Container Platform 集群。

2.1. 策略参数定义

您可以在 **BuildStrategy** 或 **ClusterBuildStrategy** 自定义资源(CR)中定义策略参数，并设置或修改，这些参数的值在 **Build** 或 **BuildRun** CR 中。您还可以在创建构建策略时配置或修改策略参数。

在为您的策略定义参数前请考虑以下点：

- 在构建策略 CR 的 **spec.parameters** 字段中定义参数列表。每个列表项都包含数组类型的名称、描述、类型和可选默认值或值。如果没有设置默认值，则必须在 **Build** 或 **BuildRun** CR 中定义值。
- 在构建策略的 **spec.steps** 字段中定义字符串或数组类型的参数。
- 使用 **$(params.your-parameter-name)** 语法指定字符串类型的参数。您可以为引用您的策略的 **Build** 或 **BuildRun** CR 中的 **your-parameter-name** 参数设置一个值。您可以根据需要定义以下字符串参数：

  表 2.1. 字符串参数

参数	描述
image	使用此参数定义自定义标签，如 **golang:$(params.go-version)**
args	使用此参数将数据传递给构建器命令
env	使用此参数为环境变量提供值

- 使用 **$(params.your-array-parameter-name[*])** 语法指定数组类型的参数。指定阵列后，您可以在参数或命令中使用它。对于数组中的每个项目，将设置一个参数。以下示例使用构建策略的 **spec.steps** 字段中的 array 参数：

```
apiVersion: shipwright.io/v1beta1
kind: ClusterBuildStrategy
metadata:
  name: <cluster_build_strategy_name>
  # ...
spec:
  parameters:
    - name: tool-args
      description: Parameters for the tool
      type: array
  steps:
    - name: a-step
```

```
command:
  - some-tool
args:
  - --tool-args
  - $(params.tool-args[*])
```

- 提供参数值作为简单字符串，或作为配置映射或 secret 中键的引用。对于参数，只有在 **spec.steps** 字段的命令、**args** 或 **env** 部分中定义时才可以使用配置映射或 secret 值。

2.2. 系统参数定义

在定义构建策略步骤来访问系统信息或 Build Run 自定义资源(CR)中的用户定义的信息时，您可以使用系统参数。您无法配置或修改系统参数，因为它们在运行时由构建运行控制器定义。

您可以在构建策略定义中定义以下系统参数：

表 2.2. 系统参数

参数	描述
$(params.shp-source-root)	表示包含源代码的目录的绝对路径。
$(params.shp-source-context)	表示源代码的上下文目录的绝对路径。如果您没有在 **Build** CR 中为 **spec.source.contextDir** 指定任何值，则此参数使用 **$(params.shp-source-root)** 系统参数的值。
$(params.shp-output-image)	表示要推送 **Build** 或 **BuildRun** CR 的 **spec.output.image** 字段中定义的镜像的 URL。

2.3. 步骤资源定义

您可以包括资源的定义，如构建策略中所有步骤对 CPU、内存和磁盘用量的限制。对于具有多个步骤的策略，步骤可能需要的资源多于其他步骤。作为策略管理员，您可以定义对每个步骤的最佳资源值。

例如，您可以使用相同的步骤安装策略，但在集群中使用不同的名称和步骤资源，以便用户可以创建具有较小的或更大资源要求的构建。

2.3.1. 使用不同资源的策略

定义多个同一策略类型，但资源的不同限制。以下示例使用与为资源定义的小和中等限制相同的 buildah 策略。这些示例提供了对步骤资源定义的更多控制。

2.3.1.1. Buildah 策略具有小限制

为 buildah 策略定义带有小资源限制的 spec.steps[].resources 字段，如下例所示：

示例：带有小限制的 buildah 策略

```
apiVersion: shipwright.io/v1beta1
kind: ClusterBuildStrategy
metadata:
  name: buildah-small
spec:
  steps:
    - name: build-and-push
      image: quay.io/containers/buildah:v1.31.0
      workingDir: $(params.shp-source-root)
      securityContext:
        capabilities:
          add:
          - "SETFCAP"
      command:
        - /bin/bash
      args:
        - -c
        - |
          set -euo pipefail
          # Parse parameters
          # ...
          # That's the separator between the shell script and its args
          - --
          - --context
          - $(params.shp-source-context)
          - --dockerfile
          - $(build.dockerfile)
          - --image
          - $(params.shp-output-image)
          - --build-args
          - $(params.build-args[*])
          - --registries-block
          - $(params.registries-block[*])
          - --registries-insecure
          - $(params.registries-insecure[*])
          - --registries-search
          - $(params.registries-search[*])
      resources:
        limits:
          cpu: 250m
          memory: 65Mi
        requests:
          cpu: 250m
```

```
        memory: 65Mi
  parameters:
    - name: build-args
      description: "The values for the args in the Dockerfile. Values must be in the format
KEY=VALUE."
      type: array
      defaults: []
    # ...
```

2.3.1.2. Buildah 策略带有介质限制

使用 buildah 策略的中等资源限值定义 spec.steps[].resources 字段，如下例所示：

示例：使用中等限制的 buildah 策略

```
apiVersion: shipwright.io/v1beta1
kind: ClusterBuildStrategy
metadata:
  name: buildah-medium
spec:
  steps:
    - name: build-and-push
      image: quay.io/containers/buildah:v1.31.0
      workingDir: $(params.shp-source-root)
      securityContext:
        capabilities:
          add:
          - "SETFCAP"
      command:
        - /bin/bash
      args:
        - -c
        - |
          set -euo pipefail
          # Parse parameters
          # ...
          # That's the separator between the shell script and its args
          - --
          - --context
          - $(params.shp-source-context)
          - --dockerfile
          - $(build.dockerfile)
          - --image
          - $(params.shp-output-image)
          - --build-args
          - $(params.build-args[*])
          - --registries-block
```

```
      - $(params.registries-block[*])
      - --registries-insecure
      - $(params.registries-insecure[*])
      - --registries-search
      - $(params.registries-search[*])
    resources:
      limits:
        cpu: 500m
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 1Gi
  parameters:
    - name: build-args
      description: "The values for the args in the Dockerfile. Values must be in the format
KEY=VALUE."
      type: array
      defaults: []
    # ...
```

为策略配置资源定义后，您必须在 Build CR 中引用策略，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-medium
spec:
  source:
    git:
      url: https://github.com/shipwright-io/sample-go
    contextDir: docker-build
  strategy:
    name: buildah-medium
    kind: ClusterBuildStrategy
  # ...
```

2.3.2. Tekton 管道中的资源管理

构建控制器可用于 Tekton 管道控制器，以便它能够调度 pod 来执行策略步骤。在运行时，构建控制器会创建一个 Tekton TaskRun 资源，而 TaskRun 资源会在特定命名空间中创建新 pod。然后，此 pod 会按顺序执行构建镜像的所有策略步骤。

2.4. 注解定义

您可以针对任何其他 Kubernetes 对象定义构建策略或集群构建策略的注解。构建策略首先将注解传播

到 TaskRun 资源。然后，Tekton 将它们传播到 pod。

您可以将注解用于以下目的：

- 要限制 pod 允许使用的网络带宽，kubernetes.io/ingress-bandwidth 和 kubernetes.io/egress-bandwidth 注解会在 Kubernetes 网络流量 shaping 功能中定义。
- 要定义容器的 AppArmor 配置集，会使用 container. apparmor.security.beta.kubernetes.io/<container_name& gt; 注解。

以下示例显示了在构建策略中使用注解：

```
apiVersion: shipwright.io/v1beta1
kind: ClusterBuildStrategy
metadata:
  name: <cluster_build_strategy_name>
  annotations:
    container.apparmor.security.beta.kubernetes.io/step-build-and-push: unconfined
    container.seccomp.security.alpha.kubernetes.io/step-build-and-push: unconfined
spec:
  # ...
```

以下注解不会被传播：

- kubectl.kubernetes.io/last-applied-configuration
- clusterbuildstrategy.shipwright.io/*
- buildstrategy.shipwright.io/*
- build.shipwright.io/*
- buildrun.shipwright.io/*

策略管理员可以使用策略引擎进一步限制注解的使用。

2.5. 安全引用字符串参数

当您在 BuildStrategy 或 ClusterBuildStrategy 自定义资源(CR)中定义环境变量、参数或镜像时，会使用字符串参数。在构建策略步骤中，您可以使用 $(params.your-parameter-name) 语法引用字符串参数。

注意

您还可以在构建策略步骤中使用 $(params.your-parameter-name) 语法来引用系统参数和策略参数。

在 pod 中，所有 $(params.your-parameter-name) 变量都由实际字符串替代。但是，在使用内联脚本引用参数中的 string 参数时，您必须注意。例如，要将参数值安全地传递给使用脚本定义的参数，您可以选择以下方法之一：

- 使用环境变量
- use 参数

示例：引用字符串参数到环境变量中

您可以将 string 参数传递给环境变量，而不是直接在脚本内使用它。通过使用环境变量的引用，您可以避免命令注入漏洞。您可以使用此方法进行策略，如 buildah。以下示例使用脚本中的环境变量来引用字符串参数：

```
apiVersion: shipwright.io/v1beta1
kind: BuildStrategy
metadata:
  name: sample-strategy
spec:
  parameters:
    - name: sample-parameter
      description: A sample parameter
      type: string
  steps:
    - name: sample-step
      env:
        - name: PARAM_SAMPLE_PARAMETER
          value: $(params.sample-parameter)
```

```
    command:
      - /bin/bash
    args:
      - -c
      - |
        set -euo pipefail

        some-tool --sample-argument "${PARAM_SAMPLE_PARAMETER}"
```

示例：引用字符串参数到参数

您可以将 string 参数传递给脚本中定义的参数。适当的 shell 引用保护命令注入.您可以使用此方法进行策略，如 buildah。以下示例使用脚本中定义的参数来引用字符串参数：

```
apiVersion: shipwright.io/v1beta1
kind: BuildStrategy
metadata:
  name: sample-strategy
spec:
  parameters:
    - name: sample-parameter
      description: A sample parameter
      type: string
  steps:
    - name: sample-step
      command:
        - /bin/bash
      args:
        - -c
        - |
          set -euo pipefail

          SAMPLE_PARAMETER="$1"

          some-tool --sample-argument "${SAMPLE_PARAMETER}"
        - --
        - $(params.sample-parameter)
```

2.6. 系统结果定义

您可以将构建策略创建的镜像的大小和摘要存储在一组结果文件中。当 BuildRun 资源失败时，您还可以存储用于调试目的的错误详情。您可以在 BuildStrategy 或 ClusterBuildStrategy CR 中定义以下结果参数：

表 2.3. 结果参数

参数	描述
$(results.shp-image-digest.path)	表示存储镜像摘要的文件的路径。

参数	描述
$(results.shp-image-size.path)	表示存储镜像的压缩大小的文件的路径。
$(results.shp-error-reason.path)	表示存储错误原因的文件的路径。
$(results.shp-error-message.path)	表示存储错误消息的文件的路径。

以下示例显示了 BuildRun CR 的 .status.output 字段中镜像的大小和摘要：

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
# ...
status:
 # ...
  output:
    digest: sha256:07626e3c7fdd28d5328a8d6df8d29cd3da760c7f5e2070b534f9b880ed093a53
    size: 1989004
  # ...
```

以下示例显示了 BuildRun CR 的 .status.failureDetails 字段中的错误原因和消息：

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
# ...
status:
  # ...
  failureDetails:
    location:
      container: step-source-default
      pod: baran-build-buildrun-gzmv5-b7wbf-pod-bbpqr
    message: The source repository does not exist, or you have insufficient permission
      to access it.
    reason: GitRemotePrivate
```

2.7. 卷和卷挂载定义

构建策略包括卷和卷挂载的定义。构建策略中定义的卷支持所有常见的 volumeSource 类型。构建步骤通过创建卷来引用卷。

注意

构建步骤中定义的卷挂载允许您访问 BuildStrategy、Build 或 Build Run 资源中定义的卷。

构建策略中的卷使用 overridable 布尔值标记，默认设置为 false。如果 Build 或 BuildRun 资源试图覆盖 BuildStrategy 资源中定义的卷，它将会失败，因为 overridable 标志的默认值为 false。

以下示例显示了定义 volumes 和 volumeMounts 字段的 BuildStrategy 资源：

```
apiVersion: shipwright.io/v1beta1
kind: BuildStrategy
metadata:
  name: buildah
spec:
  steps:
    - name: build
      image: quay.io/containers/buildah:v1.23.3
      # ...
      volumeMounts:
        - name: varlibcontainers
          mountPath: /var/lib/containers
  volumes:
    - name: varlibcontainers
      overridable: true
      emptyDir: {}
```

第 3 章 配置构建运行

BuildRun 自定义资源(CR)可帮助您定义构建引用、构建规格、参数值、服务帐户、输出、保留参数和卷来配置构建运行。BuildRun 资源可用于命名空间。

要配置构建运行，请创建一个 BuildRun 资源 YAML 文件，并将其应用到 OpenShift Container Platform 集群。

3.1. 构建中的可配置字段运行

您可以使用 BuildRun 自定义资源(CR)中的以下字段：

表 3.1. BuildRun CR 中的字段

字段	存在	描述
apiVersion	必填	指定资源的 API 版本。例如，shipped **wright.io/v1beta1**。
kind	必填	指定资源的类型。例如，**BuildRun**。
metadata	必填	表示标识自定义资源定义实例的元数据。例如，**BuildRun** 资源的名称。
spec.build.name	选填	指定要使用的现有 **Build** 资源实例。您不能将此字段与 **spec.build.spec** 字段一起使用。
spec.build.spec	选填	指定要使用的嵌入式 **构建** 资源实例。您不能将此字段与 **spec.build.name** 字段一起使用。
spec.serviceAccount	选填	指明构建镜像时要使用的服务帐户。
spec.timeout	选填	定义自定义超时。此字段值覆盖 **Build** 资源中定义的 **spec.timeout** 字段的值。
spec.paramValues	选填	表示 name-value 列表，用于指定构建策略中定义的参数值。参数值覆盖了 **Build** 资源中使用相同名称定义的参数值。
spec.output.image	选填	指明推送生成的镜像的自定义位置。此字段值覆盖 **Build** 资源中定义的 **output.image** 字段的值。
spec.output.pushSecret	选填	表示用于访问容器 registry 的现有 secret。此 secret 将添加到服务帐户中，以及 **Build** 资源请求的其他 secret。

字段	存在	描述
spec.env	选填	定义您可以传递给构建容器的额外环境变量。此字段值覆盖 **Build** 资源中指定的任何环境变量。可用的变量取决于您的构建策略使用的工具。

注意

您不能在同一 CR 中使用 spec.build.name 和 spec.build.spec 字段，因为它们是相互排斥的。

3.2. 构建引用定义

您可以在 BuildRun 资源中配置 spec.build.name 字段，以引用指示镜像构建的 Build 资源。以下示例显示了配置 spec.build.name 字段的 BuildRun CR：

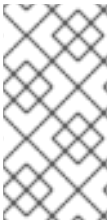
```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: buildah-buildrun
spec:
  build:
    name: buildah-build
```

3.3. 构建规格定义

您可以使用 spec.build.spec 字段将完整的构建规格嵌入到 BuildRun 资源中。通过嵌入规格，您可以在不创建和维护专用构建自定义资源的情况下构建镜像。以下示例显示了配置 spec.build.spec 字段的 BuildRun CR：

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: standalone-buildrun
spec:
  build:
    spec:
      source:
        git:
          url: https://github.com/shipwright-io/sample-go.git
        contextDir: source-build
      strategy:
        kind: ClusterBuildStrategy
```

```
    name: buildah
  output:
    image: <path_to_image>
```



注意

您不能在同一 CR 中使用 spec.build.name 和 spec.build.spec 字段，因为它们是相互排斥的。

3.4. 构建运行的参数值定义

您可以在 BuildRun CR 中为构建策略参数指定值。如果您为 Build 资源中也定义了相同名称的参数值，则 Build Run 资源中定义的值具有优先权。

在以下示例中，BuildRun 资源中的 cache 参数的值会覆盖 cache 参数的值，该值在 Build 资源中定义：

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: <your_build>
  namespace: <your_namespace>
spec:
  paramValues:
  - name: cache
    value: disabled
  strategy:
    name: <your_strategy>
    kind: ClusterBuildStrategy
  source:
  # ...
  output:
  # ...
```

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: <your_buildrun>
  namespace: <your_namespace>
spec:
  build:
    name: <your_build>
  paramValues:
  - name: cache
    value: registry
```

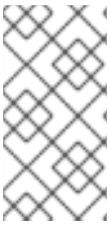
3.5. 服务帐户定义

您可以在 BuildRun 资源中定义服务帐户。服务帐户托管 Build 资源中引用的所有 secret，如下例所示：

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: buildah-buildrun
spec:
  build:
    name: buildah-build
  serviceAccount: pipeline 1
```

1

您还可以将 spec.serviceAccount 字段的值设置为 ".generate"，以便在运行时生成服务帐户。生成的服务帐户的名称与 BuildRun 资源的名称对应。



注意

当您没有定义服务帐户时，BuildRun 资源会使用 pipeline 服务帐户（如果命名空间中存在）。否则，BuildRun 资源使用 默认服务帐户。

3.6. 构建运行的保留参数定义

您可以指定 BuildRun 资源中存在已完成的构建运行的持续时间。保留参数提供了一种自动清理 BuildRun 实例的方法。您可以在 BuildRun CR 中设置以下保留参数的值：

- retention.ttlAfterFailed ：指定构建运行失败的持续时间
- retention.ttlAfterSucceeded: 指定成功构建运行可以存在的持续时间

以下示例演示了如何在 BuildRun CR 中定义保留参数：

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: buidrun-retention-ttl
spec:
  build:
    name: build-retention-ttl
```

```
  retention:
    ttlAfterFailed: 10m
    ttlAfterSucceeded: 10m
```

注意

如果您在 BuildRun 和 Build CR 中定义了 retention 参数，则 BuildRun CR 中定义的值会覆盖 Build CR 中定义的 retention 参数的值。

3.7. 构建运行的卷定义

您可以在 BuildRun CR 中定义卷。定义的卷覆盖 BuildStrategy 资源中指定的卷。如果卷没有被覆盖，则构建运行会失败。

如果 Build 和 BuildRun 资源覆盖同一卷，则 BuildRun 资源中定义的卷用于覆盖。

以下示例显示了使用 volumes 字段的 BuildRun CR：

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: <buildrun_name>
spec:
  build:
    name: <build_name>
  volumes:
    - name: <volume_name>
      configMap:
        name: <configmap_name>
```

3.8. 环境变量定义

您可以根据您的需要，在 BuildRun CR 中使用环境变量。以下示例演示了如何定义环境变量：

示例：使用环境变量定义 BuildRun 资源

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: buildah-buildrun
spec:
```

```
  build:
    name: buildah-build
  env:
    - name: <example_var_1>
      value: "<example_value_1>"
    - name: <example_var_2>
      value: "<example_value_2>"
```

以下示例显示了使用 Kubernetes Downward API 将 pod 公开为环境变量的 BuildRun 资源：

示例：定义一个 BuildRun 资源，将 pod 公开为环境变量

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: buildah-buildrun
spec:
  build:
    name: buildah-build
  env:
    - name: <pod_name>
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
```

以下示例显示了使用 Kubernetes Downward API 将容器作为环境变量公开的 BuildRun 资源：

示例：定义一个 BuildRun 资源，将容器作为环境变量公开

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: buildah-buildrun
spec:
  build:
    name: buildah-build
  env:
    - name: MEMORY_LIMIT
```

```
    valueFrom:
      resourceFieldRef:
        containerName: <my_container>
        resource: limits.memory
```

3.9. 构建运行状态

每当镜像构建状态更改时，BuildRun 资源都会更新，如下例所示：

示例：带有 Unknown 状态的 BuildRun

```
$ oc get buildrun buildah-buildrun-mp99r
NAME                  SUCCEEDED   REASON    STARTTIME   COMPLETIONTIME
buildah-buildrun-mp99r  Unknown     Unknown     1s
```

示例：带有 True 状态的 BuildRun

```
$ oc get buildrun buildah-buildrun-mp99r
NAME                  SUCCEEDED   REASON    STARTTIME   COMPLETIONTIME
buildah-buildrun-mp99r  True        Succeeded     29m       20m
```

BuildRun 资源在 status.conditions 字段中存储与状态相关的信息。例如，类型为 Succeeded 的条件表示资源已成功完成其操作。status.conditions 字段包含 BuildRun 资源的重要信息，如 status、reason 和 message。

3.9.1. 构建运行状态描述

BuildRun 自定义资源(CR)在镜像构建过程中可能会具有不同的状态。下表描述了构建运行的不同状态：

表 3.2. 构建运行的状态

Status	原因	描述
Unknown	**待处理**	**BuildRun** 资源会等待状态为 **Pending** 的 pod。
Unknown	**运行中**	**BuildRun** 资源已被验证并启动，以执行其工作。
Unknown	**BuildRunCanceled**	用户已请求取消构建运行。此请求会触发构建运行控制器，以发出取消相关任务运行的请求。当存在此状态时，取消仍然处于进程中。
True	**Succeeded**	**BuildRun** 资源的 pod 已创建。
False	**Failed**	**BuildRun** 资源的步骤之一失败。
False	**BuildRunTimeout**	**BuildRun** 资源的执行超时。
False	**UnknownStrategyKind**	**Kind** 字段中定义的策略类型未知。您可以定义这些策略类型： **ClusterBuildStrategy** 和 **BuildStrategy**。
False	**ClusterBuildStrategyNotFound**	集群中没有找到引用的集群范围的策略。
False	**BuildStrategyNotFound**	集群中没有找到引用的命名空间范围的策略。
False	**SetOwnerReferenceFailed**	将 **BuildRun** 资源的 **ownerReferences** 字段设置为相关的 **TaskRun** 资源会失败。
False	**TaskRunIsMissing**	未找到与 **BuildRun** 资源相关的 **TaskRun** 资源。
False	**TaskRunGenerationFailed**	**TaskRun** 规格的生成失败。
False	**MissingParameterValues**	您没有为构建策略中定义的一些参数提供任何值，而无需任何默认值。您必须在 **Build** 或 **BuildRun** CR 中提供这些参数的值。
False	**RestrictedParametersInUse**	提供了 system 参数的值，这是不允许的。
False	**UndefinedParameter**	提供了构建策略中没有定义的参数值。
False	**WrongParameterValueType**	为带有错误类型的构建策略参数提供了一个值。例如，如果参数定义为构建策略中的数组或字符串，您必须相应地提供一组值或直接值。
False	**InconsistentParameterValues**	参数的值包含多个这些值： **value**、**configMapValue** 和 **secretValue**。您必须仅提供上述值之一才能保持一致性。

Status	原因	描述
False	**EmptyArrayItemParameterValues**	数组参数的值中的项目不包含这些值：**value**、**configMapValue** 和 **secretValue**。您必须仅提供上述值之一，因为不允许 null 数组项。
False	**IncompleteConfigMapValueParameterValues**	参数的值包含一个 **configMapValue** 值，其中 **name** 或 **value** 字段为空。您必须指定空字段以指向命名空间中的现有配置映射键。
False	**IncompleteSecretValueParameterValues**	参数的值包含一个 **secretValue** 值，其中 **name** 或 **value** 字段为空。您必须指定空字段以指向命名空间中的现有 secret 密钥。
False	**ServiceAccountNotFound**	在集群中没有找到引用的服务帐户。
False	**BuildRegistrationFailed**	**BuildRun** 资源中引用的构建处于 **Failed** 状态。
False	**BuildNotFound**	**BuildRun** 资源中引用的构建没有找到。
False	**BuildRunCanceled**	**BuildRun** 和相关的 **TaskRun** 资源已被成功取消。
False	**BuildRunNameInvalid**	**metadata.name** 字段中定义的构建运行名称无效。您必须在 **BuildRun** CR 中为构建运行名称提供有效的标签值。
False	**BuildRunNoRefOrSpec**	**BuildRun** 资源没有定义 **spec.build.name** 或 **spec.build.spec** 字段。
False	**BuildRunAmbiguousBuild**	定义的 **BuildRun** 资源使用 **spec.build.name** 和 **spec.build.spec** 字段。一次只允许其中一个参数。
False	**BuildRunBuildFieldOverrideForbidden**	定义的 **spec.build.name** 字段与 **spec.build.spec** 字段结合使用一个覆盖，这是不允许的。使用 **spec.build.spec** 字段直接指定对应的值。
False	**PodEvicted**	构建运行 pod 从其运行的节点驱除。

3.9.2. 失败的构建运行

当构建运行时，您可以检查 BuildRun CR 中的 status.failureDetails 字段，以识别 pod 或容器中发生故障的确切点。status.failureDetails 字段包含错误消息以及失败原因。只有在构建策略中定义时，您才会看到消息和失败原因。

以下示例显示了失败的构建运行：

```
# ...
status:
  # ...
  failureDetails:
    location:
      container: step-source-default
      pod: baran-build-buildrun-gzmv5-b7wbf-pod-bbpqr
    message: The source repository does not exist, or you have insufficient permission
      to access it.
    reason: GitRemotePrivate
```

注意

status.failureDetails 字段还为与 Git 相关的所有操作提供错误详情。

3.9.3. 步骤结果为构建运行状态

在 BuildRun 资源完成其执行后，.status 字段包含从构建运行控制器生成的步骤发出的 .status.taskResults 结果。结果包括用于构建镜像的源代码的镜像摘要或提交 SHA。在 BuildRun 资源中，.status.sources 字段包含执行源步骤的结果，.status.output 字段包含执行输出步骤的结果。

以下示例显示了带有 Git 源步骤结果的 BuildRun 资源：

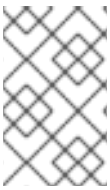
示例：带有 Git 源步骤结果的 BuildRun 资源

```
# ...
status:
  buildSpec:
    # ...
  output:
    digest: sha256:07626e3c7fdd28d5328a8d6df8d29cd3da760c7f5e2070b534f9b880ed093a53
    size: 1989004
  sources:
  - name: default
    git:
      commitAuthor: xxx xxxxxx
      commitSha: f25822b85021d02059c9ac8a211ef3804ea8fdde
      branchName: main
```

以下示例显示了带有本地源代码步骤结果的 BuildRun 资源：

示例：带有本地源代码的步骤结果的 BuildRun 资源

```
# ...
status:
  buildSpec:
    # ...
  output:
    digest: sha256:07626e3c7fdd28d5328a8d6df8d29cd3da760c7f5e2070b534f9b880ed093a53
    size: 1989004
  sources:
  - name: default
    bundle:
      digest: sha256:0f5e2070b534f9b880ed093a537626e3c7fdd28d5328a8d6df8d29cd3da760c7
```



注意

只有在构建策略中定义时，才会查看输出镜像的摘要和大小。

3.9.4. 构建快照

对于每个构建运行协调，如果现有任务运行是该构建运行的一部分，则 BuildRun 资源更新状态中的 buildSpec 字段。

在这个版本中，Build 资源快照会生成并嵌入到 BuildRun 资源的 status.buildSpec 字段中。因此，buildSpec 字段包含原始 构建 规格的确切副本，后者用于执行特定的镜像构建。通过使用构建快照，您可以看到原始的 Build 资源配置。

3.10. 使用 TEKTON 任务运行的构建关系

BuildRun 资源将镜像构造的任务委托给 Tekton TaskRun 资源，后者运行所有步骤，直到任务完成，或者在任务中发生失败。

在构建运行协调过程中，构建运行控制器会生成一个新的 TaskRun 资源。控制器在 TaskRun 资源中嵌入构建执行所需的步骤。嵌入式步骤在构建策略中定义。

3.11. 构建可取消运行

您可以通过将状态设置为 BuildRun Canceled 来取消活跃的 BuildRun 实例。当您取消 BuildRun 实例时，底层 TaskRun 资源也会标记为已取消。

以下示例显示了 BuildRun 资源的已取消构建运行：

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: buildah-buildrun
spec:
  # [...]
  state: "BuildRunCanceled"
```

3.12. 自动构建运行删除

要自动删除构建运行，您可以在 build 或 build run 规格中添加以下保留参数：

- buildrun TTL 参数： 确保构建仅在完成后的定义持续时间内都存在。
  - buildrun.spec.retention.ttlAfterFailed: 如果指定时间已通过，则构建运行会被删除，构建运行失败。
  - buildrun.spec.retention.ttlAfterSucceeded: 如果指定时间已通过，则构建运行会被删除，构建运行成功。
- 构建 TTL 参数：确保构建仅在完成后的定义持续时间内仅存在构建。
  - build.spec.retention.ttlAfterFailed ：如果指定时间已通过，则构建运行会被删除，并且构建运行失败。
  - build.spec.retention.ttlAfterSucceeded: 如果指定时间已通过，并且构建运行成功，则构建运行会被删除。
- 构建 限制参数： 确保构建只能存在有限数量的成功或失败的构建。

- **build.spec.retention.succeededLimit ：定义构建可以存在的成功构建数量。**

- **build.spec.retention.failedLimit ：定义构建可以存在的失败构建数量。**