builds for Red Hat OpenShift 1.0

Configure

Configuring Builds

Last Updated: 2024-02-13

builds for Red Hat OpenShift 1.0 Configure

Configuring Builds

Abstract

This document provides information about configuring Builds.

Table of Contents

CHAPTER 1. CONFIGURING BUILDS

In a **Build** custom resource (CR), you can define the source, build strategy, parameter values, output, retention parameters, and volumes to configure a build. A **Build** resource is available for use within a namespace.

For configuring a build, create a **Build** resource YAML file and apply it to the OpenShift Container Platform cluster.

1.1. CONFIGURABLE FIELDS IN BUILD

You can use the following fields in your **Build** custom resource (CR):

Table 1.1. Fields in the**Build** CR

Field	Presence	Description
apiVersion	Required	Specifies the API version of the resource, for example, **shipwright.io/v1beta1**.
kind	Required	Specifies the type of the resource, for example, **Build**.
metadata	Required	Denotes the metadata that identifies the custom resource definition instance, for example, the name of the **Build** resource.
spec.source	Required	Denotes the location of the source code, for example, a Git repository or source bundle image.
spec.strategy	Required	Denotes the name and type of the strategy used for the **Build** resource.
spec.output	Required	Denotes the location where the generated image will be pushed.
spec.output.pushSecret	Required	Denotes an existing secret to get access to the container registry.
spec.paramValues	Optional	Denotes a name-value list to specify values for parameters defined in the build strategy.
spec.timeout	Optional	Defines a custom timeout. The default value is ten minutes. You can overwrite this field value in your **BuildRun** resource.
spec.output.annotations	Optional	Denotes a list of key-value pair that you can use to annotate the output image.
spec.output.labels	Optional	Denotes a list of key-value pair that you can use to label the output image.

Field	Presence	Description
spec.env	Optional	Defines additional environment variables that you can pass to the build container. The available variables depend on the tool that is used by your build strategy.
spec.retention.ttlAfterFailed	Optional	Specifies the duration for which a failed build run can exist.
spec.retention.ttlAfterSucceeded	Optional	Specifies the duration for which a successful build run can exist.
spec.retention.failedLimit	Optional	Specifies the number of failed build runs that can exist.
spec.retention.succeededLimit	Optional	Specifies the number of successful build runs that can exist.

1.2. SOURCE DEFINITION

You can configure the source details for a build in the **Build** custom resource (CR) by setting the value of the following fields:

- **source.git.url**: Defines the source location of the image available in a Git repository.
- **source.git.cloneSecret**: References a secret in the namespace that contains the SSH private key for a private Git repository.
- **source.git.revision**: Defines a specific revision to select from the source Git repository. For example, a commit, tag, or branch name. This field defaults to the Git repository default branch.
- **source.contextDir**: Specifies the context path for the repositories where the source code is not present at the root folder.

The build controller does not automatically validate that the Git repository you specified for pulling an image exists. If you need to validate, set the value of the **build.shipwright.io/verify.repository** annotation to **true**, as shown in the following example:

```
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
```

The build controller validates the existence of a Git repository in the following scenarios:

- When you use the endpoint URL with an HTTP or HTTPS protocol.

- When you have defined an SSH protocol, such as **git@**, but not a referenced secret, such as **source.git.cloneSecret**.

The following examples show how you can configure a build with different set of source inputs.

Example: Configuring a build with credentials

You can configure a build with a source by specifying your credentials, as shown in the following example:

```
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-build
spec:
  source:
    git:
      url: https://github.com/sclorg/nodejs-ex
      cloneSecret: source-repository-credentials
```

Example: Configuring a build with a context path

You can configure a build with a source that specifies a context path in the Git repository, as shown in the following example:

```
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
```

Example: Configuring a build with a tag

You can configure a build with a source that specifies the tag **v.0.1.0** for the Git repository, as shown in the following example:

```
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
```

Example: Configuring a build with environment variables

You can also configure a build that specifies environment variables, as shown in the following example:

```
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. STRATEGY DEFINITION

You can configure the strategy for a build in the **Build** CR. The following build strategies are available for use:

- **buildah**
- **source-to-image**

To configure a build strategy, define the **spec.strategy.name** and **spec.strategy.kind** fields in the **Build** CR, as shown in the following example:

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

1.4. PARAMETER VALUES DEFINITION FOR A BUILD

You can specify values for the build strategy parameters in your **Build** CR. By specifying parameter values, you can control how the steps of the build strategy work. You can also overwrite the values in the **BuildRun** resource.

For all parameters, you must specify values either directly or by using reference keys from config maps or secrets.

NOTE

The usage of the parameter in the build strategy steps limits the usage of config maps and secrets. You can only use config maps and secrets if the parameter is used in the command, argument, or environment variable.

When using the **paramValues** field in your **Build** CR, avoid the following scenarios:

- Specifying a **spec.paramValues** name that does not match one of the **spec.parameters** defined in the **BuildStrategy** CR.

- Specifying a **spec.paramValues** name that collides with the Shipwright reserved parameters. These parameters include **BUILDER_IMAGE**, **CONTEXT_DIR**, and any name starting with **shp-**.

Also, ensure that you understand the content of your strategy before defining the **paramValues** field in the **Build** CR.

1.4.1. Example configuration for defining parameter values

The following examples show how to define parameters in a build strategy and assign values to those parameters by using a **Build** CR. You can also assign a value to a parameter of the type **array** in your **Build** CR.

Example: Defining parameters in a ClusterBuildStrategy CR

The following example shows a **ClusterBuildStrategy** CR that defines several parameters:

```
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:
# ...
```

Example: Assigning values to parameters in a Build CR

The above **ClusterBuildStrategy** CR defines a **storage-driver** parameter and you can specify the value of the **storage-driver** parameter in your **Build** CR, as shown in the following example:

```
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:
  # ...
```

Example: Creating a **ConfigMap** CR to control a parameter centrally

If you want to use the **storage-driver** parameter for multiple builds and control its usage centrally, then you can create a **ConfigMap** CR, as shown in the following example:

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

You can use the created **ConfigMap** CR as a parameter value in your **Build** CR, as shown in the following example:

```
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:
  # ...
```

Example: Assigning value to a parameter of the type **array** in a **Build** CR

You can assign value to a parameter of the type **array**. If you use the **buildah** strategy, you can define a **registries-search** parameter to search images in specific registries. The following example shows how you can assign a value to the **registries-search** array parameter:

```
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:
  # ...
```

Example: Referencing a secret in a Build CR

You can reference a secret for a **registries-block** array parameter, as shown in the following example:

```
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 The value references a secret.

1.5. BUILDER OR DOCKER FILE DEFINITION

In your **Build** CR, you can use the **spec.paramValues** field to specify the image that contains the tools to build the output image. The following example specifies a **Dockerfile** image in a **Build** CR:

```
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
```

You can also use a **builder** image as part of the **source-to-image** build strategy in your **Build** CR, as shown in the following example:

```
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. OUTPUT DEFINITION

In your **Build** CR, you can specify an output location to push the image. When using an external private registry as your output location, you must specify a secret to access the image. You can also specify the annotations and labels for the output image.

NOTE

When you specify annotations or labels, the output image is pushed twice. The first push comes from the build strategy and the second push changes the image configuration to add the annotations and labels.

The following example defines a public registry where the image is pushed:

```
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
```

The following example defines a private registry where the image is pushed:

```
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
```

The following example defines annotations and labels for the image:

```
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. RETENTION PARAMETERS DEFINITION FOR A BUILD

You can define retention parameters for the following purposes:

- To specify how long a completed build run can exist
- To specify the number of succeeded or failed build runs that can exist for a build

Retention parameters provide a way to clean your **BuildRun** instances or resources automatically. You can set the value of the following retention parameters in your **Build** CR:

- **retention.succeededLimit**: Defines the number of succeeded build runs that can exist for a build.
- **retention.failedLimit**: Defines the number of failed build runs that can exist for a build.
- **retention.ttlAfterFailed**: Specifies the duration for which a failed build run can exist.
- **retention.ttlAfterSucceeded**: Specifies the duration for which a successful build run can exist.

The following example shows the usage of retention parameters in a **Build** CR:

```
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
  # ...
```

NOTE

When you change the value of the **retention.failedLimit** and **retention.succeededLimit** parameters, the new limit is enforced as soon as those changes are applied on your build. However, when you change the value of the **retention.ttlAfterFailed** and **retention.ttlAfterSucceeded** parameters, the new retention duration is enforced only on the new build runs. Old build runs adhere to the old retention duration. If you have defined retention duration in both **BuildRun** and **Build** CRs, the retention duration defined in the **BuildRun** CR gets the priority.

1.8. VOLUMES DEFINITION FOR A BUILD

You can define volumes in your **Build** CR. The defined volumes override the volumes specified in the **BuildStrategy** resource. If a volume is not overridden, then the build run fails.

The following example shows the usage of the **volumes** field in a **Build** CR:

```
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>
```

CHAPTER 2. CONFIGURING BUILD STRATEGIES

In a **BuildStrategy** or **ClusterBuildStrategy** custom resource (CR), you can define strategy parameters, system parameters, step resources definitions, annotations, and volumes to configure a build strategy. A **BuildStrategy** resource is available for use within a namespace, and a **ClusterBuildStrategy** resource is available for use throughout the cluster.

To configure a build strategy, create a **BuildStrategy** or **ClusterBuildStrategy** resource YAML file and apply it to the OpenShift Container Platform cluster.

2.1. STRATEGY PARAMETERS DEFINITION

You can define strategy parameters in a **BuildStrategy** or **ClusterBuildStrategy** custom resource (CR) and set, or modify, the values of those parameters in your **Build** or **BuildRun** CR. You can also configure or modify strategy parameters at build time when creating your build strategy.

Consider the following points before defining parameters for your strategy:

- Define a list of parameters in the **spec.parameters** field of your build strategy CR. Each list item contains a name, a description, a type, and an optional default value, or values, for an array type. If no default value is set, you must define a value in the **Build** or **BuildRun** CR.
- Define parameters of string or array type in the **spec.steps** field of your build strategy.
- Specify a parameter of string type by using the **$(params.your-parameter-name)** syntax. You can set a value for the **your-parameter-name** parameter in your **Build** or **BuildRun** CR that references your strategy. You can define the following string parameters based on your needs:

  Table 2.1. String parameters

Parameter	Description
image	Use this parameter to define a custom tag, such as **golang:$(params.go-version)**
args	Use this parameter to pass data into your builder commands
env	Use this parameter to provide a value for an environment variable

- Specify a parameter of array type by using the **$(params.your-array-parameter-name[*])** syntax. After specifying the array, you can use it in an argument or a command. For each item in the array, an argument will be set. The following example uses an array parameter in the **spec.steps** field of the build strategy:

```
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[*])
```

- Provide parameter values as simple strings or as references to keys in config maps or secrets. For a parameter, you can use a config map or secret value only if it is defined in the **command**, **args**, or **env** section of the **spec.steps** field.

2.2. SYSTEM PARAMETERS DEFINITION

You can use system parameters when defining the steps of a build strategy to access system information, or user-defined information in a **Build** or **BuildRun** custom resource (CR). You cannot configure or modify system parameters as they are defined at runtime by the build run controller.

You can define the following system parameters in your build strategy definition:

Table 2.2. System parameters

Parameter	Description
$(params.shp-source-root)	Denotes the absolute path to the directory that contains the source code.
$(params.shp-source-context)	Denotes the absolute path to the context directory of the source code. If you do not specify any value for **spec.source.contextDir** in the **Build** CR, this parameter uses the value of the **$(params.shp-source-root)** system parameter.
$(params.shp-output-image)	Denotes the URL of the image to push as defined in the **spec.output.image** field of your **Build** or **BuildRun** CR.

2.3. STEP RESOURCES DEFINITION

You can include a definition of resources, such as the limit imposed on CPU, memory, and disk usage for all steps in a build strategy. For strategies with multiple steps, a step might require more resources than others. As a strategy administrator, you can define the resource values that are optimal for each step.

For example, you can install strategies with the same steps, but different names and step resources on the cluster so that users can create a build with smaller or larger resource requirements.

2.3.1. Strategies with different resources

Define multiple types of the same strategy with varying limits on the resources. The following examples use the same **buildah** strategy with small and medium limits defined for the resources. These examples provide a strategy administrator more control over the step resources definition.

2.3.1.1. Buildah strategy with small limit

Define the **spec.steps[].resources** field with a small resource limit for the **buildah** strategy, as shown in the following example:

Example: buildah strategy with small limit

```
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 strategy with medium limit

Define the **spec.steps[].resources** field with a medium resource limit for the **buildah** strategy, as shown in the following example:

Example: **buildah** strategy with medium limit

```
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: []
    # ...
```

After configuring the resource definition for a strategy, you must reference the strategy in your **Build** CR, as shown in the following example:

```
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. Resource management in Tekton pipelines

The build controller works with the Tekton pipeline controller so that it can schedule pods for executing the strategy steps. At runtime, the build controller creates a Tekton **TaskRun** resource, and the **TaskRun** resource creates a new pod in the specific namespace. This pod then sequentially executes all of the strategy steps to build an image.

2.4. ANNOTATIONS DEFINITION

You can define annotations for a build strategy or a cluster build strategy like for any other Kubernetes object. The build strategy first propagates annotations to the **TaskRun** resource. Then, Tekton propagates them to the pod.

You can use annotations for the following purposes:

- To limit the network bandwidth the pod is allowed to use, the **kubernetes.io/ingress-bandwidth** and **kubernetes.io/egress-bandwidth** annotations are defined in the Kubernetes network traffic shaping feature.
- To define the AppArmor profile of a container, the **container.apparmor.security.beta.kubernetes.io/<container_name>** annotation is used.

The following example shows the usage of annotations in a build strategy:

```
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:
  # ...
```

The following annotations are not propagated:

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

A strategy administrator can further restrict the usage of annotations by using policy engines.

2.5. SECURE REFERENCING OF STRING PARAMETERS

String parameters are used when you define environment variables, arguments, or images in a **BuildStrategy** or **ClusterBuildStrategy** custom resource (CR). In your build strategy steps, you can reference string parameters by using the **$(params.your-parameter-name)** syntax.

NOTE

You can also reference system parameters and strategy parameters by using the **$(params.your-parameter-name)** syntax in your build strategy steps.

In the pod, all **$(params.your-parameter-name)** variables are replaced by actual strings. However, you must pay attention when you reference a string parameter in an argument by using an inline script. For example, to securely pass a parameter value into an argument defined with a script, you can choose one of the following approaches:

- Use environment variables
- Use arguments

Example: Referencing a string parameter into an environment variable

You can pass the string parameter into an environment variable, instead of directly using it inside the script. By using quoting around the environment variable, you can avoid the command injection vulnerability. You can use this approach for strategies, such as **buildah**. The following example uses an environment variable inside the script to reference a string parameter:

```
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}"
```

Example: Referencing a string parameter into an argument

You can pass the string parameter into an argument defined within your script. Appropriate shell quoting guards against command injection. You can use this approach for strategies, such as **buildah**. The following example uses an argument defined within your script to reference a string parameter:

```
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. SYSTEM RESULTS DEFINITION

You can store the size and digest of the image that is created by your build strategy to a set of result files. You can also store error details for debugging purposes when a **BuildRun** resource fails. You can define the following result parameters in your **BuildStrategy** or **ClusterBuildStrategy** CR:

Table 2.3. Result parameters

Parameter	Description
$(results.shp-image-digest.path)	Denotes the path to the file that stores the digest of the image.
$(results.shp-image-size.path)	Denotes the path to the file that stores the compressed size of the image.
$(results.shp-error-reason.path)	Denotes the path to the file that stores the error reason.
$(results.shp-error-message.path)	Denotes the path to the file that stores the error message.

The following example shows the size and digest of the image in the **.status.output** field of the **BuildRun** CR:

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

The following example shows the error reason and message in the **.status.failureDetails** field of the **BuildRun** CR:

```
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. VOLUMES AND VOLUME MOUNTS DEFINITION

A build strategy includes the definition of volumes and volume mounts. The volumes defined in a build strategy support all of the usual **volumeSource** types. The build steps refer to the volumes by creating a volume mount.

NOTE

The volume mount defined in build steps allows you to access volumes defined in a **BuildStrategy**, **Build** or **BuildRun** resource.

Volumes in build strategy use an **overridable** boolean flag, which is set to **false** by default. If a **Build** or **BuildRun** resource tries to override the volumes defined in a **BuildStrategy** resource, it will fail because the default value of the **overridable** flag is **false**.

The following example shows a **BuildStrategy** resource that defines the **volumes** and **volumeMounts** fields:

```
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: {}
```

CHAPTER 3. CONFIGURING BUILD RUNS

In a **BuildRun** custom resource (CR), you can define the build reference, build specification, parameter values, service account, output, retention parameters, and volumes to configure a build run. A **BuildRun** resource is available for use within a namespace.

For configuring a build run, create a **BuildRun** resource YAML file and apply it to the OpenShift Container Platform cluster.

3.1. CONFIGURABLE FIELDS IN BUILD RUN

You can use the following fields in your **BuildRun** custom resource (CR):

Table 3.1. Fields in the BuildRun CR

Field	Presence	Description
apiVersion	Required	Specifies the API version of the resource. For example, **shipwright.io/v1beta1**.
kind	Required	Specifies the type of the resource. For example, **BuildRun**.
metadata	Required	Indicates the metadata that identifies the custom resource definition instance. For example, the name of the **BuildRun** resource.
spec.build.name	Optional	Specifies an existing **Build** resource instance to use. You cannot use this field with the **spec.build.spec** field.
spec.build.spec	Optional	Specifies an embedded **Build** resource instance to use. You cannot use this field with the **spec.build.name** field.
spec.serviceAccount	Optional	Indicates the service account to use when building the image.
spec.timeout	Optional	Defines a custom timeout. This field value overwrites the value of the **spec.timeout** field defined in your **Build** resource.
spec.paramValues	Optional	Indicates a name-value list to specify values for parameters defined in the build strategy. The parameter value overwrites the value of the parameter that is defined with the same name in your **Build** resource.
spec.output.image	Optional	Indicates a custom location where the generated image will be pushed. This field value overwrites the value of the **output.image** field defined in your **Build** resource.

Field	Presence	Description
spec.output.pushSecret	Optional	Indicates an existing secret to get access to the container registry. This secret will be added to the service account along with other secrets requested by the **Build** resource.
spec.env	Optional	Defines additional environment variables that you can pass to the build container. This field value overrides any environment variables that are specified in the **Build** resource. The available variables depend on the tool that is used by your build strategy.

NOTE

You cannot use the **spec.build.name** and **spec.build.spec** fields together in the same CR because they are mutually exclusive.

3.2. BUILD REFERENCE DEFINITION

You can configure the **spec.build.name** field in your **BuildRun** resource to reference a **Build** resource that indicates an image to build. The following example shows a **BuildRun** CR that configures the **spec.build.name** field:

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

3.3. BUILD SPECIFICATION DEFINITION

You can embed a complete build specification into your **BuildRun** resource using the **spec.build.spec** field. By embedding specifications, you can build an image without creating and maintaining a dedicated **Build** custom resource. The following example shows a **BuildRun** CR that configures the **spec.build.spec** field:

```
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>
```

NOTE

You cannot use the **spec.build.name** and **spec.build.spec** fields together in the same CR because they are mutually exclusive.

3.4. PARAMETER VALUES DEFINITION FOR A BUILD RUN

You can specify values for the build strategy parameters in your **BuildRun** CR. If you have provided a value for a parameter that is also defined in the **Build** resource with the same name, then the value defined in the **BuildRun** resource takes priority.

In the following example, the value of the **cache** parameter in the **BuildRun** resource overrides the value of the **cache** parameter, which is defined in the **Build** resource:

```
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. SERVICE ACCOUNT DEFINITION

You can define a service account in your **BuildRun** resource. The service account hosts all secrets referenced in your **Build** resource, as shown in the following example:

```
apiVersion: shipwright.io/v1beta1
```

```
kind: BuildRun
metadata:
  name: buildah-buildrun
spec:
  build:
    name: buildah-build
  serviceAccount: pipeline (1)
```

(1) You can also set the value of the **spec.serviceAccount** field to **".generate"** to generate the service account during runtime. The name of the generated service account corresponds with the name of the **BuildRun** resource.

NOTE

When you do not define the service account, the **BuildRun** resource uses the **pipeline** service account if it exists in the namespace. Otherwise, the **BuildRun** resource uses the **default** service account.

3.6. RETENTION PARAMETERS DEFINITION FOR A BUILD RUN

You can specify the duration for which a completed build run can exist in your **BuildRun** resource. Retention parameters provide a way to clean your **BuildRun** instances automatically. You can set the value of the following retention parameters in your **BuildRun** CR:

- **retention.ttlAfterFailed**: Specifies the duration for which a failed build run can exist
- **retention.ttlAfterSucceeded**: Specifies the duration for which a successful build run can exist

The following example shows how to define retention parameters in your **BuildRun** CR:

```
apiVersion: shipwright.io/v1beta1
kind: BuildRun
metadata:
  name: buidrun-retention-ttl
spec:
  build:
    name: build-retention-ttl
  retention:
    ttlAfterFailed: 10m
    ttlAfterSucceeded: 10m
```

NOTE

If you have defined a retention parameter in both **BuildRun** and **Build** CRs, the value defined in the **BuildRun** CR overrides the value of the retention parameter defined in the **Build** CR.

3.7. VOLUMES DEFINITION FOR A BUILD RUN

You can define volumes in your **BuildRun** CR. The defined volumes override the volumes specified in the **BuildStrategy** resource. If a volume is not overridden, then the build run fails.

In case the **Build** and **BuildRun** resources override the same volume, the volume defined in the **BuildRun** resource is used for overriding.

The following example shows a **BuildRun** CR that uses the **volumes** field:

```
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. ENVIRONMENT VARIABLES DEFINITION

You can use environment variables in your **BuildRun** CR based on your needs. The following example shows how to define environment variables:

Example: Defining a BuildRun resource with environment variables

```
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>"
```

The following example shows a **BuildRun** resource that uses the Kubernetes downward API to expose a pod as an environment variable:

Example: Defining a BuildRun resource to expose a pod as an environment variable

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

The following example shows a **BuildRun** resource that uses the Kubernetes downward API to expose a container as an environment variable:

Example: Defining a **BuildRun** resource to expose a container as an environment variable

```
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. BUILD RUN STATUS

The **BuildRun** resource updates whenever the image building status changes, as shown in the following examples:

Example: BuildRun with Unknown status

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

Example: BuildRun with True status

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

A **BuildRun** resource stores the status-related information in the **status.conditions** field. For example, a condition with the type **Succeeded** indicates that resources have successfully completed their operation. The **status.conditions** field includes significant information like status, reason, and message for the **BuildRun** resource.

3.9.1. Build run statuses description

A **BuildRun** custom resource (CR) can have different statuses during the image building process. The following table covers the different statuses of a build run:

Table 3.2. Statuses of a build run

Status	Cause	Description
Unknown	**Pending**	The **BuildRun** resource waits for a pod in status **Pending**.

Status	Cause	Description
Unknown	**Running**	The **BuildRun** resource has been validated and started to perform its work.
Unknown	**BuildRunCanceled**	The user has requested to cancel the build run. This request triggers the build run controller to make a request for canceling the related task runs. Cancellation is still under process when this status is present.
True	**Succeeded**	The pod for the **BuildRun** resource is created.
False	**Failed**	The **BuildRun** resource is failed in one of the steps.
False	**BuildRunTimeout**	The execution of the **BuildRun** resource is timed out.
False	**UnknownStrategyKind**	The strategy type defined in the **Kind** field is unknown. You can define these strategy types: **ClusterBuildStrategy** and **BuildStrategy**.
False	**ClusterBuildStrategyNotFound**	The referenced cluster-scoped strategy was not found in the cluster.
False	**BuildStrategyNotFound**	The referenced namespace-scoped strategy was not found in the cluster.
False	**SetOwnerReferenceFailed**	Setting the **ownerReferences** field from the **BuildRun** resource to the related **TaskRun** resource failed.
False	**TaskRunIsMissing**	The **TaskRun** resource related to the **BuildRun** resource was not found.
False	**TaskRunGenerationFailed**	The generation of a **TaskRun** specification has failed.
False	**MissingParameterValues**	You have not provided any value for some parameters that are defined in the build strategy without any default. You must provide the values for those parameters in the **Build** or the **BuildRun** CR.
False	**RestrictedParametersInUse**	A value for a system parameter was provided, which is not allowed.
False	**UndefinedParameter**	A value for a parameter was provided that is not defined in the build strategy.

Status	Cause	Description
False	**WrongParameterValueType**	A value was provided for a build strategy parameter with the wrong type. For example, if the parameter is defined as an array or a string in the build strategy, you must provide a set of values or a direct value accordingly.
False	**InconsistentParameterValues**	A value for a parameter contained more than one of these values: **value**, **configMapValue**, and **secretValue**. You must provide only one of the mentioned values to maintain consistency.
False	**EmptyArrayItemParameterValues**	An item inside the values of an array parameter contained none of these values: **value**,**configMapValue**, and **secretValue**. You must provide only one of the mentioned values as null array items are not allowed.
False	**IncompleteConfigMapValueParameterValues**	A value for a parameter contained a **configMapValue** value where the **name** or the **value** field was empty. You must specify the empty field to point to an existing config map key in your namespace.
False	**IncompleteSecretValueParameterValues**	A value for a parameter contained a **secretValue** value where the **name** or the **value** field was empty. You must specify the empty field to point to an existing secret key in your namespace.
False	**ServiceAccountNotFound**	The referenced service account was not found in the cluster.
False	**BuildRegistrationFailed**	The referenced build in the **BuildRun** resource is in a **Failed** state.
False	**BuildNotFound**	The referenced build in the **BuildRun** resource was not found.
False	**BuildRunCanceled**	The **BuildRun** and related **TaskRun** resources were canceled successfully.
False	**BuildRunNameInvalid**	The defined build run name in the **metadata.name** field is invalid. You must provide a valid label value for the build run name in your **BuildRun** CR.
False	**BuildRunNoRefOrSpec**	The **BuildRun** resource does not have either the **spec.build.name** or **spec.build.spec** field defined.
False	**BuildRunAmbiguousBuild**	The defined **BuildRun** resource uses both the **spec.build.name** and **spec.build.spec** fields. Only one of the parameters is allowed at a time.

Status	Cause	Description
False	**BuildRunBuildFieldOverrideForbidden**	The defined **spec.build.name** field uses an override in combination with the **spec.build.spec** field, which is not allowed. Use the **spec.build.spec** field to directly specify the respective value.
False	**PodEvicted**	The build run pod was evicted from the node it was running on.

3.9.2. Failed build runs

When a build run fails, you can check the **status.failureDetails** field in your **BuildRun** CR to identify the exact point where the failure happened in the pod or container. The **status.failureDetails** field includes an error message and a reason for the failure. You only see the message and reason for failure if they are defined in your build strategy.

The following example shows a failed build run:

```
# ...
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
```

NOTE

The **status.failureDetails** field also provides error details for all operations related to Git.

3.9.3. Step results in build run status

After a **BuildRun** resource completes its execution, the **.status** field contains the **.status.taskResults** result emitted from the steps generated by the build run controller. The result includes the image digest or the commit SHA of the source code that is used for building the image. In a **BuildRun** resource, the **.status.sources** field contains the result from the execution of source steps and the **.status.output** field contains the result from the execution of output steps.

The following example shows a **BuildRun** resource with step results for a Git source:

Example: A BuildRun resource with step results for a Git source

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

The following example shows a **BuildRun** resource with step results for a local source code:

Example: A BuildRun resource with step results for a local source code

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

NOTE

You get to see the digest and size of the output image only if it is defined in your build strategy.

3.9.4. Build snapshot

For each build run reconciliation, the **buildSpec** field in the status of the **BuildRun** resource updates if an existing task run is part of that build run.

During this update, a **Build** resource snapshot generates and embeds into the **status.buildSpec** field of the **BuildRun** resource. Due to this, the **buildSpec** field contains an exact copy of the original **Build** specification, which was used to execute a particular image build. By using the build snapshot, you can see the original **Build** resource configuration.

3.10. RELATIONSHIP OF BUILD RUN WITH TEKTON TASKS

The **BuildRun** resource delegates the task of image construction to the Tekton **TaskRun** resource, which runs all steps until either the completion of the task, or a failure occurs in the task.

During the build run reconciliation, the build run controller generates a new **TaskRun** resource. The controller embeds the required steps for a build run execution in the **TaskRun** resource. The embedded steps are defined in your build strategy.

3.11. BUILD RUN CANCELLATION

You can cancel an active **BuildRun** instance by setting its state to **BuildRunCanceled**. When you cancel a **BuildRun** instance, the underlying **TaskRun** resource is also marked as canceled.

The following example shows a canceled build run for a **BuildRun** resource:

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

3.12. AUTOMATIC BUILD RUN DELETION

To automatically delete a build run, you can add the following retention parameters in the **build** or **buildrun** specification:

- **buildrun** TTL parameters: Ensures that build runs only exist for a defined duration of time after completion.
  - **buildrun.spec.retention.ttlAfterFailed**: The build run is deleted if the specified time has passed and the build run has failed.
  - **buildrun.spec.retention.ttlAfterSucceeded**: The build run is deleted if the specified time has passed and the build run has succeeded.
- **build** TTL parameters: Ensures that build runs for a build only exist for a defined duration of time after completion.
  - **build.spec.retention.ttlAfterFailed**: The build run is deleted if the specified time has passed and the build run has failed for the build.
  - **build.spec.retention.ttlAfterSucceeded**: The build run is deleted if the specified time has passed and the build run has succeeded for the build.
- **build** limit parameters: Ensures that only a limited number of succeeded or failed build runs can exist for a build.
  - **build.spec.retention.succeededLimit**: Defines the number of succeeded build runs that can exist for the build.
  - **build.spec.retention.failedLimit**: Defines the number of failed build runs that can exist for the build.