Red Hat
OpenShift

Builds and Image Streams

Builds

A *build* is the process of transforming input parameters into a resulting object. Most often, the process is used to transform input parameters or source code into a runnable image. A `BuildConfig` object is the definition of the entire build process.

OpenShift Container Platform leverages Kubernetes by creating Docker-formatted containers from build images and pushing them to a container image registry.

Build objects share common characteristics: inputs for a build, the need to complete a build process, logging the build process, publishing resources from successful builds, and publishing the final status of the build. Builds take advantage of resource restrictions, specifying limitations on resources such as CPU usage, memory usage, and build or pod execution time.

The OpenShift Container Platform build system provides extensible support for *build strategies* that are based on selectable types specified in the build API. There are three primary build strategies available:

- Docker build
- Source-to-Image (S2I) build
- Custom build

By default, Docker builds and S2I builds are supported.

The resulting object of a build depends on the builder used to create it. For Docker and S2I builds, the resulting objects are runnable images. For Custom builds, the resulting objects are whatever the builder image author has specified.

Additionally, the Pipeline build strategy can be used to implement sophisticated workflows:

- continuous integration
- continuous deployment

For a list of build commands, see the Developer's Guide.

For more information on how OpenShift Container Platform leverages Docker for builds, see the upstream documentation.

Docker Build

The Docker build strategy invokes the docker build command, and it therefore expects a repository with a ***Dockerfile*** and all required artifacts in it to produce a runnable image.

Source-to-Image (S2I) Build

Source-to-Image (S2I) is a tool for building reproducible, Docker-formatted container images. It produces ready-to-run images by injecting application source into a container image and assembling a new image. The new image incorporates the base image (the builder) and built source and is ready to use with the `docker run` command. S2I supports incremental builds, which re-use previously downloaded dependencies, previously built artifacts, etc.

The advantages of S2I include the following:

Image flexibility	S2I scripts can be written to inject application code into almost any existing Docker-formatted container image, taking advantage of the existing ecosystem. Note that, currently, S2I relies on `tar` to inject application source, so the image needs to be able to process tarred content.
Speed	With S2I, the assemble process can perform a large number of complex operations without creating a new layer at each step, resulting in a fast process. In addition, S2I scripts can be written to re-use artifacts stored in a previous version of the application image, rather than having to download or build them each time the build is run.
Patchability	S2I allows you to rebuild the application consistently if an underlying image needs a patch due to a security issue.
Operational efficiency	By restricting build operations instead of allowing arbitrary actions, as a ***Dockerfile*** would allow, the PaaS operator can avoid accidental or intentional abuses of the build system.
Operational security	Building an arbitrary ***Dockerfile*** exposes the host system to root privilege escalation. This can be exploited by a malicious user because the entire Docker build process is run as a user with Docker privileges. S2I restricts the operations performed as a root user and can run the scripts as a non-root user.
User efficiency	S2I prevents developers from performing arbitrary `yum install` type operations, which could slow down development iteration, during their application build.
Ecosystem	S2I encourages a shared ecosystem of images where you can leverage best practices for your applications.
Reproducibility	Produced images can include all inputs including specific versions of build tools and dependencies. This ensures that the image can be reproduced precisely.

Custom Build

The Custom build strategy allows developers to define a specific builder image responsible for the entire build process. Using your own builder image allows you to customize your build process.

A Custom builder image is a plain Docker-formatted container image embedded with build process logic, for example for building RPMs or base images. The `openshift/origin-custom-docker-builder` image is available on the Docker Hub registry as an example implementation of a Custom builder image.

Pipeline Build

The Pipeline build strategy allows developers to define a *Jenkins pipeline* for execution by the Jenkins pipeline plugin. The build can be started, monitored, and managed by OpenShift Container Platform in the same way as any other build type.

Pipeline workflows are defined in a Jenkinsfile, either embedded directly in the build configuration, or supplied in a Git repository and referenced by the build configuration.

The first time a project defines a build configuration using a Pipeline strategy, OpenShift Container Platform instantiates a Jenkins server to execute the pipeline. Subsequent Pipeline build configurations in the project share this Jenkins server.

For more details on how the Jenkins server is deployed and how to configure or disable the autoprovisioning behavior, see Configuring Pipeline Execution.

> The Jenkins server is not automatically removed, even if all Pipeline build configurations are deleted. It must be manually deleted by the user.

For more information about Jenkins Pipelines, see the Jenkins documentation.

Image Streams

An image stream and its associated tags provide an abstraction for referencing container images from within OpenShift Container Platform. The image stream and its tags allow you to see what images are available and ensure that you are using the specific image you need even if the image in the repository changes.

Image streams do not contain actual image data, but present a single virtual view of related images, similar to an image repository.

You can configure Builds and Deployments to watch an image stream for notifications when new images are added and react by performing a Build or Deployment, respectively.

For example, if a Deployment is using a certain image and a new version of that image is created, a Deployment could be automatically performed to pick up the new version of the image.

However, if the image stream tag used by the Deployment or Build is not updated, then even if the container image in the container image registry is updated, the Build or Deployment will continue using the previous (presumably known good) image.

The source images can be stored in any of the following:

- OpenShift Container Platform's integrated registry
- An external registry, for example `registry.redhat.io` or `hub.docker.com`
- Other image streams in the OpenShift Container Platform cluster

When you define an object that references an image stream tag (such as a Build or Deployment configuration), you point to an image stream tag, not the Docker repository. When you Build or Deploy your application, OpenShift Container Platform queries the Docker repository using the image stream tag to locate the associated ID of the image and uses that exact image.

The image stream metadata is stored in the etcd instance along with other cluster information.

The following image stream contains two tags: `34` which points to a Python v3.4 image and `35` which points to a Python v3.5 image:

```
$ oc describe is python
```

Example Output

```
Name:                   python
Namespace:              imagestream
Created:                25 hours ago
Labels:                 app=python
Annotations:            openshift.io/generated-by=OpenShiftWebConsole
                        openshift.io/image.dockerRepositoryCheck=2017-10-03T19:48:00Z
Docker Pull Spec:       docker-registry.default.svc:5000/imagestream/python
Image Lookup:           local=false
Unique Images:          2
Tags:                   2

34
  tagged from centos/python-34-centos7

  * centos/python-34-
centos7@sha256:28178e2352d31f240de1af1370be855db33ae9782de737bb005247d8791a54d0
      14 seconds ago

35
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:2efb79ca3ac9c9145a63675fb0c09220ab3b8d4005d35e0644417ee552548b10
      7 seconds ago
```

Using image streams has several significant benefits:

- You can tag, rollback a tag, and quickly deal with images, without having to re-push using the command line.
- You can trigger Builds and Deployments when a new image is pushed to the registry. Also, OpenShift Container Platform has generic triggers for other resources (such as Kubernetes objects).
- You can mark a tag for periodic re-import. If the source image has changed, that change is picked up and reflected in the image stream, which triggers the Build and/or Deployment flow, depending upon the Build or Deployment configuration.
- You can share images using fine-grained access control and quickly distribute images across your teams.
- If the source image changes, the image stream tag will still point to a known-good version of the image, ensuring that your application will not break unexpectedly.
- You can configure security around who can view and use the images through permissions on the image stream objects.
- Users that lack permission to read or list images on the cluster level can still retrieve the images tagged in a project using image streams.

For a curated set of image streams, see the OpenShift Image Streams and Templates library.

When using image streams, it is important to understand what the image stream tag is pointing to and how changes to tags and images can affect you. For example:

- If your image stream tag points to a container image tag, you need to understand how that container image tag is updated. For example, a container image tag `docker.io/ruby:2.5` points to a v2.5 ruby image, but a container image tag `docker.io/ruby:latest` changes with major versions. So, the container image tag that a image stream tag points to can tell you how stable the image stream tag is.
- If your image stream tag follows another image stream tag instead of pointing directly to a container image tag, it is possible that the image stream tag might be updated to follow a different image stream tag in the future. This change might result in picking up an incompatible version change.

Important terms

Docker repository
A collection of related container images and tags identifying them. For example, the OpenShift Jenkins images are in a Docker repository:

```
docker.io/openshift/jenkins-2-centos7
```

Container registry
A content server that can store and service images from Docker repositories. For example:

```
registry.redhat.io
```

container image
A specific set of content that can be run as a container. Usually associated with a particular tag within a Docker repository.

container image tag
A label applied to a container image in a repository that distinguishes a specific image. For example, here `3.6.0` is a tag:

```
docker.io/openshift/jenkins-2-centos7:3.6.0
```

> A container image tag can be updated to point to new container image content at any time.

container image ID
A SHA (Secure Hash Algorithm) code that can be used to pull an image. For example:

```
docker.io/openshift/jenkins-2-centos7@sha256:ab312bda324
```

> A SHA image ID cannot change. A specific SHA identifier always references the exact same container image content.

Image stream
An OpenShift Container Platform object that contains pointers to any number of Docker-formatted container images identified by tags. You can think of an image stream as equivalent to a Docker repository.

Image stream tag
A named pointer to an image in an image stream. An image stream tag is similar to a container image tag. See Image Stream Tag below.

Image stream image
An image that allows you to retrieve a specific container image from a particular image stream where it is tagged. An image stream image is an API resource object that pulls together some metadata about a particular image SHA identifier. See Image Stream Images below.

Image stream trigger
A trigger that causes a specific action when an image stream tag changes. For example, importing can cause the value of the tag to change, which causes a trigger to fire when there are Deployments, Builds, or other resources listening for those. See Image Stream Triggers below.

Configuring Image Streams

An image stream object file contains the following elements.

> See the Developer Guide for details on managing images and image streams.

Image Stream Object Definition

```
apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  creationTimestamp: 2017-09-29T13:33:49Z
  generation: 1
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample  (1)
  namespace: test
  resourceVersion: "633"
  selflink: /oapi/v1/namespaces/test/imagestreams/origin-ruby-sample
  uid: ee2b9405-c68c-11e5-8a99-525400f25e34
spec: {}
status:
  dockerImageRepository: 172.30.56.218:5000/test/origin-ruby-sample  (2)
  tags:
  - items:
    - created: 2017-09-02T10:15:09Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d  (3)
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
 (4)
    - created: 2017-09-29T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
      generation: 1
      image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
    tag: latest  (5)
```

1 The name of the image stream.

2 Docker repository path where new images can be pushed to add/update them in this image stream.

3 The SHA identifier that this image stream tag currently references. Resources that reference this image stream tag use this identifier.

4 The SHA identifier that this image stream tag previously referenced. Can be used to rollback to an older image.

5 The image stream tag name.

For a sample build configuration that references an image stream, see What Is a BuildConfig? in the `Strategy` stanza of the configuration.

For a sample deployment configuration that references an image stream, see Creating a Deployment Configuration in the `Strategy` stanza of the configuration.

Image Stream Images

An *image stream image* points from within an image stream to a particular image ID.

Image stream images allow you to retrieve metadata about an image from a particular image stream where it is tagged.

Image stream image objects are automatically created in OpenShift Container Platform whenever you import or tag an image into the image stream. You should never have to explicitly define an image stream image object in any image stream definition that you use to create image streams.

The image stream image consists of the image stream name and image ID from the repository, delimited by an `@` sign:

```
<image-stream-name>@<image-id>
```

To refer to the image in the image stream object example above, the image stream image looks like:

```
origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

Image Stream Tags

An *image stream tag* is a named pointer to an image in an *image stream*. It is often abbreviated as *istag*. An image stream tag is used to reference or retrieve an image for a given image stream and tag.

Image stream tags can reference any local or externally managed image. It contains a history of images represented as a stack of all images the tag ever pointed to. Whenever a new or existing image is tagged under particular image stream tag, it is placed at the first position in the history stack. The image previously occupying the top position will be available at the second position, and so forth. This allows for easy rollbacks to make tags point to historical images again.

The following image stream tag is from the image stream object example above:

Image Stream Tag with Two Images in its History

```
  tags:
  - items:
    - created: 2017-09-02T10:15:09Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
    - created: 2017-09-29T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
      generation: 1
      image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
    tag: latest
```

Image stream tags can be *permanent* tags or *tracking* tags.

- *Permanent tags* are version-specific tags that point to a particular version of an image, such as Python 3.5.
- *Tracking tags* are reference tags that follow another image stream tag and could be updated in the future to change which image they follow, much like a symlink. Note that these new levels are not guaranteed to be backwards-compatible.

  For example, the `latest` image stream tags that ship with OpenShift Container Platform are tracking tags. This means consumers of the `latest` image stream tag will be updated to the newest level of the framework provided by the image when a new level becomes available. A `latest` image stream tag to `v3.10` could be changed to `v3.11` at any time. It is important to be aware that these `latest` image stream tags behave differently than the Docker `latest` tag. The `latest` image stream tag, in this case, does not point to the latest image in the Docker repository. It points to another image stream tag, which might not be the latest version of an image. For example, if the `latest` image stream tag points to `v3.10` of an image, when the `3.11` version is released, the `latest` tag is not automatically updated to `v3.11`, and remains at `v3.10` until it is manually updated to point to a `v3.11` image stream tag.

  Tracking tags are limited to a single image stream and cannot reference other image streams.

You can create your own image stream tags for your own needs. See the Recommended Tagging Conventions.

The image stream tag is composed of the name of the image stream and a tag, separated by a colon:

```
<image stream name>:<tag>
```

For example, to refer to the `sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d` image in the image stream object example above, the image stream tag would be:

```
origin-ruby-sample:latest
```

Image Stream Change Triggers

Image stream triggers allow your Builds and Deployments to be automatically invoked when a new version of an upstream image is available.

For example, Builds and Deployments can be automatically started when an image stream tag is modified. This is achieved by monitoring that particular image stream tag and notifying the Build or Deployment when a change is detected.

The `ImageChange` trigger results in a new replication controller whenever the content of an image stream tag changes (when a new version of the image is pushed).

An ImageChange Trigger

```
triggers:
  - type: "ImageChange"
    imageChangeParams:
      automatic: true  (1)
      from:
        kind: "ImageStreamTag"
        name: "origin-ruby-sample:latest"
        namespace: "myproject"
      containerNames:
        - "helloworld"
```

1 If the `imageChangeParams.automatic` field is set to `false`, the trigger is disabled.

With the above example, when the `latest` tag value of the **origin-ruby-sample** image stream changes and the new image value differs from the current image specified in the deployment configuration's **helloworld** container, a new replication controller is created using the new image for the **helloworld** container.

> If an `ImageChange` trigger is defined on a deployment configuration (with a `ConfigChange` trigger and `automatic=false`, or with `automatic=true`) and the `ImageStreamTag` pointed by the `ImageChange` trigger does not exist yet, then the initial deployment process will automatically start as soon as an image is imported or pushed by a build to the `ImageStreamTag`.

Image Stream Mappings

When the integrated registry receives a new image, it creates and sends an image stream mapping to OpenShift Container Platform, providing the image's project, name, tag, and image metadata.

> Configuring image stream mappings is an advanced feature.

This information is used to create a new image (if it does not already exist) and to tag the image into the image stream. OpenShift Container Platform stores complete metadata about each image, such as commands, entry point, and environment variables. Images in OpenShift Container Platform are immutable and the maximum name length is 63 characters.

> See the Developer Guide for details on manually tagging images.

The following image stream mapping example results in an image being tagged as **test/origin-ruby-sample:latest**:

Image Stream Mapping Object Definition

```
apiVersion: v1
kind: ImageStreamMapping
metadata:
  creationTimestamp: null
  name: origin-ruby-sample
  namespace: test
tag: latest
image:
  dockerImageLayers:
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ee1dd2cb6df21971f4af6de0f1d7782b81fb63156801cfde2bb47b4247c23c29
    size: 196634330
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ca062656bff07f18bff46be00f40cfbb069687ec124ac0aa038fd676cfaea092
    size: 177723024
  - name: sha256:63d529c59c92843c395befd065de516ee9ed4995549f8218eac6ff088bfa6b6e
    size: 55679776
  - name: sha256:92114219a04977b5563d7dff71ec4caa3a37a15b266ce42ee8f43dba9798c966
    size: 11939149
  dockerImageMetadata:
    Architecture: amd64
    Config:
      Cmd:
      - /usr/libexec/s2i/run
      Entrypoint:
      - container-entrypoint
      Env:
      - RACK_ENV=production
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - OPENSHIFT_BUILD_NAME=ruby-sample-build-1
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
      ExposedPorts:
        8080/tcp: {}
      Labels:
        build-date: 2015-12-23
        io.k8s.description: Platform for building and running Ruby 2.2 applications
        io.k8s.display-name: 172.30.56.218:5000/test/origin-ruby-sample:latest
        io.openshift.build.commit.author: Ben Parees
<bparees@users.noreply.github.com>
        io.openshift.build.commit.date: Wed Jan 20 10:14:27 2016 -0500
        io.openshift.build.commit.id: 00cadc392d39d5ef9117cbc8a31db0889eedd442
        io.openshift.build.commit.message: 'Merge pull request #51 from php-
coder/fix_url_and_sti'
        io.openshift.build.commit.ref: master
        io.openshift.build.image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
        io.openshift.build.source-location: https://github.com/openshift/ruby-hello-
world.git
```

```
        io.openshift.builder-base-version: 8d95148
        io.openshift.builder-version: 8847438ba06307f86ac877465eadc835201241df
        io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
        io.openshift.tags: builder,ruby,ruby22
        io.s2i.scripts-url: image:///usr/libexec/s2i
        license: GPLv2
        name: CentOS Base Image
        vendor: CentOS
      User: "1001"
      WorkingDir: /opt/app-root/src
    Container: 86e9a4a3c760271671ab913616c51c9f3cea846ca524bf07c04a6f6c9e103a76
    ContainerConfig:
      AttachStdout: true
      Cmd:
      - /bin/sh
      - -c
      - tar -C /tmp -xf - && /usr/libexec/s2i/assemble
      Entrypoint:
      - container-entrypoint
      Env:
      - RACK_ENV=production
      - OPENSHIFT_BUILD_NAME=ruby-sample-build-1
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - PATH=/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
      ExposedPorts:
        8080/tcp: {}
      Hostname: ruby-sample-build-1-build
      Image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
      OpenStdin: true
      StdinOnce: true
      User: "1001"
      WorkingDir: /opt/app-root/src
    Created: 2016-01-29T13:40:00Z
    DockerVersion: 1.8.2.fc21
    Id: 9d7fd5e2d15495802028c569d544329f4286dcd1c9c085ff5699218dbaa69b43
    Parent: 57b08d979c86f4500dc8cad639c9518744c8dd39447c055a3517dc9c18d6fccd
    Size: 441976279
    apiVersion: "1.0"
    kind: DockerImage
  dockerImageMetadataVersion: "1.0"
  dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

Working with Image Streams

The following sections describe how to use image streams and image stream tags. For more information on working with image streams, see Managing Images.

Getting Information about Image Streams

To get general information about the image stream and detailed information about all the tags it is pointing to, use the following command:

```
$ oc describe is/<image-name>
```

For example:

```
$ oc describe is/python
```

Example Output

```
Name:                   python
Namespace:              default
Created:                About a minute ago
Labels:                 <none>
Annotations:            openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec:       docker-registry.default.svc:5000/default/python
Image Lookup:           local=false
Unique Images:          1
Tags:                   1

3.5
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      About a minute ago
```

To get all the information available about particular image stream tag:

```
$ oc describe istag/<image-stream>:<tag-name>
```

For example:

```
$ oc describe istag/python:latest
```

Example Output

```
Image Name:     sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Docker Image:   centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Name:           sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Created:        2 minutes ago
Image Size:     251.2 MB (first layer 2.898 MB, last binary layer 72.26 MB)
Image Created:  2 weeks ago
Author:         <none>
Arch:           amd64
Entrypoint:     container-entrypoint
Command:        /bin/sh -c $STI_SCRIPTS_PATH/usage
Working Dir:    /opt/app-root/src
User:           1001
Exposes Ports:  8080/tcp
Docker Labels:  build-date=20170801
```

More information is output than shown.

Adding Additional Tags to an Image Stream

To add a tag that points to one of the existing tags, you can use the `oc tag` command:

```
oc tag <image-name:tag> <image-name:tag>
```

For example:

```
$ oc tag python:3.5 python:latest
```

Example Output

```
Tag python:latest set to
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25.
```

Use the `oc describe` command to confirm the image stream has two tags, one (`3.5`) pointing at the external container image and another tag (`latest`) pointing to the same image because it was created based on the first tag.

```
$ oc describe is/python
```

Example Output

```
Name:                   python
Namespace:              default
Created:                5 minutes ago
Labels:                 <none>
Annotations:            openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec:       docker-registry.default.svc:5000/default/python
Image Lookup:           local=false
Unique Images:          1
Tags:                   2

latest
  tagged from python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      About a minute ago

3.5
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      5 minutes ago
```

Adding Tags for an External Image

Use the `oc tag` command for all tag-related operations, such as adding tags pointing to internal or external images:

```
$ oc tag <repositiory/image> <image-name:tag>
```

For example, this command maps the `docker.io/python:3.6.0` image to the `3.6` tag in the `python` image stream.

```
$ oc tag docker.io/python:3.6.0 python:3.6
```

Example Output

```
Tag python:3.6 set to docker.io/python:3.6.0.
```

If the external image is secured, you will need to create a secret with credentials for accessing that registry. See Importing Images from Private Registries for more details.

Updating an Image Stream Tag

To update a tag to reflect another tag in an image stream:

```
$ oc tag <image-name:tag> <image-name:latest>
```

For example, the following updates the `latest` tag to reflect the `3.6` tag in an image stream:

```
$ oc tag python:3.6 python:latest
```

Example Output

```
Tag python:latest set to
python@sha256:438208801c4806548460b27bd1fbcb7bb188273d13871ab43f.
```

Removing Image Stream Tags from an Image Stream

To remove old tags from an image stream:

```
$ oc tag -d <image-name:tag>
```

For example:

```
$ oc tag -d python:3.5
```

Example Output

```
Deleted tag default/python:3.5.
```

Configuring Periodic Importing of Tags

When working with an external container image registry, to periodically re-import an image (such as, to get latest security updates), use the `--scheduled` flag:

```
$ oc tag <repositiory/image> <image-name:tag> --scheduled
```

For example:

```
$ oc tag docker.io/python:3.6.0 python:3.6 --scheduled
```

Example Output

```
Tag python:3.6 set to import docker.io/python:3.6.0 periodically.
```

This command causes OpenShift Container Platform to periodically update this particular image stream tag. This period is a cluster-wide setting set to 15 minutes by default.

To remove the periodic check, re-run above command but omit the `--scheduled` flag. This will reset its behavior to default.

```
$ oc tag <repositiory/image> <image-name:tag>
```