Source to image(S2I)

Source-To-Image (S2I) is a standalone tool which is very useful when creating builder images. It also happens that S2I is the major strategy used for building applications in OpenShift 3. The main reasons one might be interested in using source builds are:

- **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.
- **Patchability** – S2I allows you to rebuild the application consistently if an underlying image needs a patch due to a security issue.
- **User efficiency** – S2I prevents developers from performing arbitrary yum install type operations during their application build, which results in slow development iteration.
- **Ecosystem** – S2I encourages a shared ecosystem of images where you can leverage best practices for your applications..
  This article is about creating a simple S2I builder image.

S2I generates a new Docker image using source code and a builder Docker image. The S2I project includes a number of commonly used Docker builder images, such as Python or Ruby, you can also extend S2I with your own custom scripts.

Before going into details of how to create a builder image, let's explain how S2I works and the role of the builder image within the build process. Source-To-Image, as the name implies, is responsible for transforming your application source into an executable Dockerimage that you can later run inside of OpenShift 3 or directly via docker run. The builder image contains the specific intelligence required to produce that executable image (also known as a build artifact). The total build process involves several steps, including the invocation of the builder image. In this article we will focus on the simplest workflow:

Download the S2I scripts (or use the one from the inside builder image).
Download the application source.
S2I then streams the scripts and application sources into the builder image container.
It then runs the assemble script, which is defined in the builder image.
Save the final image.
There is some required content of the builder image to make all this work. First of all, since the builder image is responsible for actually building the application, it has to contain all the necessary libraries and tools needed to build and run an application. For example, a Tomcat builder image will have Tomcat, the JDK, and Maven installed while a Python builder image may have Gunicorn or Cherry.py, SciPy binaries, and pip. Secondly, it needs to contain the script logic to actually perform the build and run operations. This part is handled by the two required S2I Scripts:

assemble – which is responsible for the build of an application
run – which is responsible for running the application.
In the following steps I will show you how to construct a builder image with an example docker image using the Lighttpd server that serves static html files . There's an accompanying github repository with all the files created during each step corresponding to a commit in the repository.

Let's then start our journey by getting the latest binary of S2I from https://github.com/openshift/source-to-image/releases/ (at the time of writing this article I was using version 1.0.9).

STEP 1
S2I comes with a handy command that bootstraps the required directory structure for a builder image. You can read more about it here. Let's use it to create all the necessary artifacts:

```
s2i create lighttpd-centos7 s2i-lighttpd
```

I've invoked s2i create passing the future name of the builder image (lighttpd-centos7) and the directory where the artifacts should be created (s2i-lighttpd). If the directory does not exist it'll be created for you. The result of running the above command is the following directory structure:

s2i-lighttpd/

Dockerfile – This is a standard Dockerfile where we'll define the builder image
Makefile – a helper script for building and testing the builder image
test/
  run – test script, testing if the builder image works correctly
  test-app/ – directory for your test application
  .s2i/bin/
assemble – script responsible for building the application
run – script responsible for running the application
save-artifacts – script responsible for incremental builds, covered in a future article
usage – script responsible for printing the usage of the builder image

STEP 2
We'll first start with defining how the builder image will look like by modifying the Dockerfile:

```
# We are basing our builder image on openshift base-centos7 image
FROM openshift/base-centos7

# Inform users who's the maintainer of this builder image
MAINTAINER Maciej Szulik <maszulik@redhat.com>

# Inform about software versions being used inside the builder
ENV LIGHTTPD_VERSION=1.4.35

# Set labels used in OpenShift to describe the builder images
LABEL io.k8s.description="Platform for serving static HTML files" \
io.k8s.display-name="Lighttpd 1.4.35" \
io.openshift.expose-services="8080:http" \
io.openshift.tags="builder,html,lighttpd"

# Install the required software, namely Lighttpd and
RUN yum install -y lighttpd && \
# clean yum cache files, as they are not needed and will only make the image bigger in the end
```

```
yum clean all -y

# Defines the location of the S2I
# Although this is defined in openshift/base-centos7 image it's repeated here
# to make it clear why the following COPY operation is happening
LABEL io.openshift.s2i.scripts-url=image:///usr/local/s2i
# Copy the S2I scripts from ./.s2i/bin/ to /usr/local/s2i when making the builder image
COPY ./.s2i/bin/ /usr/local/s2i

# Copy the lighttpd configuration file
COPY ./etc/ /opt/app-root/etc

# Drop the root user and make the content of /opt/app-root owned by user 1001
RUN chown -R 1001:1001 /opt/app-root

# Set the default user for the image, the user itself was created in the base image
USER 1001

# Specify the ports the final image will expose
EXPOSE 8080

# Set the default CMD to print the usage of the image, if somebody does docker run
CMD ["usage"]
```

STEP 3

Once the Dockerfile is defined we can now start to fill in the remaining parts of the builder image. Let's deal with S2I scripts. We'll start with assemble, which is responsible for building the application. In our case it'll just copy the source files into the directory from which they will be served by the Lighttpd server:

```
#!/bin/bash -e
#
# S2I assemble script for the 'lighttpd-centos7' image.
# The 'assemble' script builds your application source ready to run.
#
# For more information refer to the documentation:
# https://github.com/openshift/source-to-image/blob/master/docs/builder_image.md
#

echo "---> Installing application source"
cp -Rf /tmp/src/. ./
```

By default the s2i build places the application source in /tmp/src directory. This directory is where the source and other assets will be placed for the build process. You can modify this location by setting the io.openshift.s2i.destination label or passing --destination flag, in which case the sources will be placed in the src subdirectory of the directory you specified. The destination in the above command (./) is using working directory set in the openshift/base-centos7 image, which is set to be /opt/app-root/src.

Now it's time to handle the second required script – run, which is responsible for running the application. In our case it'll just handle starting the Lighttpd server:

```
#!/bin/bash -e
#
# S2I run script for the 'lighttpd-centos7' image.
# The run script executes the server that runs your application.
#
# For more information see the documentation:
# https://github.com/openshift/source-to-image/blob/master/docs/builder_image.md
#

exec lighttpd -D -f /opt/app-root/etc/lighttpd.conf
```

We're using exec for the above command to replace the run script's process with the Lighttpd server process. This is done so that all the signals sent by docker will go to Lighttpd process and to have the output (stdout and stderr) of Lighttpd available when running this image.

Since we are not covering incremental builds in our example, we can safely remove the save-artifacts script. Finally we put some information on how to use our builder image in the usage script:

```
#!/bin/bash -e

cat <<EOF
This is the lighttpd-centos7 S2I image:
To use it, install S2I: https://github.com/openshift/source-to-image
```

```
Sample invocation:

s2i build https://github.com/soltysh/sti-lighttpd.git --context-dir=test/test-app/ lighttpd-centos7 sample-app

You can then run the resulting image via:
docker run -p 8080:8080 sample-app
EOF
```

NOTE: Make sure to check the script file permissions; they should all be 755.

STEP 4
The last remaining piece of our builder image is the configuration file for Lighttpd. We use it in the Dockerfile and run script. Without it, the server won't work. I'm creating it in s2i-lighttpd/etc/lighttpd.conf with the following contents:

```
# directory where the documents will be served from
server.document-root = "/opt/app-root/src"

# port the server listens on
server.port = 8080

# default file if none is provided in the URL
index-file.names = ( "index.html" )

# configure specific mimetypes, otherwise application/octet-stream will be used for every file
mimetype.assign = (
".html" => "text/html",
".txt" => "text/plain",
".jpg" => "image/jpeg",
".png" => "image/png"
)
```

This is the simplest possible configuration file for Lighttpd. We're just setting:

the path from which the content will be served (/opt/app-root/src),
the port the server listens for connections on (8080),
the default file for directory (index.html),
the mimetype mappings to serve files properly.
Our builder image is ready. We can make sure it's being built properly by invoking the make command in the s2i-lighttpd directory which, as you can see from the Makefile, invokes a plain docker build command.

STEP 5
Now it's time to test our builder image with a sample application. I'm creating an index.html file in the s2i-lighttpd/test/test-app directory with the following contents:

```
<!doctype html>
<html>
<head>
<title>test-app</title>
</head>
<body>
<h1>Hello from lighttpd served index.html!</h1>
</body>
```

With this file in place, we can now do our first S2I build. Let's invoke the following command from the s2i-lighttpd directory:

```
s2i build test/test-app/ lighttpd-centos7 sample-app
```

We are building an application from the test/test-app/ directory, using our newly built image (lighttpd-centos7). The resulting image will be named sample-app. By default, the S2I build prints all the output from the assemble script, so you should see following text in the console (for now you can safely ignore warnings about using local builder image and non-git directory):

```
---> Installing application source
```

Now it's time to actually test the resulting image. Run the image, publishing port 8080 from the container to one on localhost, with the following command:

```
docker run -p 8080:8080 sample-app
```

You should now be able to visit http://localhost:8080/ and see the contents of the index.html file there.

STEP 6
There's one final element generated by the s2i create command that we didn't cover yet: the tests. Generally speaking the generated s2i-lighttpd/test/run script can be used almost as is, assuming you've chosen the default generated port in the Dockerfile. In that case, you can run the test suite with the following command from thes2i-lighttpd directory:

```
make test
```

make test
NOTE: Make sure to check the test/run file permissions; it should be 700.

This script runs the s2i build (make sure to have the s2i executable in your PATH) using test-app as an application source and then performs a series of tests to make sure the image is usable. These tests include:

checking that the s2i build finished with no errors,
checking that the usage script is working as expected,
running the resulting image,
checking that the running application container responds properly.
I did not include the contents of the test/run script here, since it's bit lengthy. Interested users are encouraged to check out the accompanying repository https://github.com/soltysh/sti-lighttpd/. The commit history shows the exact steps presented in this article.

Congratulations, you've produced an S2I-enabled builder image that can, used with S2I (iow. Source Build strategy in OpenShift), consume html files in a git repository and produce a new image that will serve those html files when run. Generalizing this, it's easy to see how you can define other builder images with appropriate assemble, run scripts that will consume other types of application source and produce images that build and run those applications.