

# Quarkus - Deploying on OpenShift with S2I

This guide covers:

- The deployment of the application to OpenShift using S2I to build

## Prerequisites

For this guide you need:

- roughly 5 minutes
- having access to an OpenShift cluster. Minishift is a valid option.

## Solution

We recommend to follow the instructions in the next sections and build the application step by step. However, you can go right to the completed example.

Clone the Git repository: `git clone https://github.com/quarkusio/quarkus-quickstarts.git`, or download an [archive](#).

The solution is located in the `getting-started` directory.

## Deploying the application as GraalVM native executable in OpenShift

In this section, we are going to leverage the S2I build mechanism of OpenShift. We use Quarkus' GraalVM Native S2I Builder, and therefore do not need a `Dockerfile` in this approach. You do not need to locally clone the Git repository, as it will be directly built inside OpenShift. We are going to create an OpenShift `build` executing it:

```

# To build the image on OpenShift
oc new-app quay.io/quarkus/ubi-quarkus-native-s2i:19.3.1-
java8~https://github.com/quarkusio/quarkus-quickstarts.git
--context-dir=getting-started --name=quarkus-quickstart-native
oc logs -f bc/quarkus-quickstart-native

# To create the route
oc expose svc/quarkus-quickstart-native

# Get the route URL
export URL="http://$(oc get route | grep quarkus-quickstart-native
| awk '{print $2}')"
echo $URL
curl $URL/hello/greeting/quarkus

```



The `oc new-app` command above uses a builder image compatible with JDK 8. In order to create a JDK 11 native compatible image use `quay.io/quarkus/ubi-quarkus-native-s2i:19.3.1-java11` as a builder image.

Your application is accessible at the printed URL.

Note that GraalVM-based native build are more memory & CPU intensive than regular pure Java builds. [By default, builds are completed by pods using unbound resources, such as memory and CPU](#), but note that [your OpenShift Project may have limit ranges defined](#).

Testing indicates that the "hello, world" getting-started demo application builds in around 2 minutes on typical hardware when the build is given 4 GB of RAM and 4 (virtual) CPUs for concurrency. You therefore may need to increase the respective limits for OpenShift's S2I build containers like so:

```

apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "quarkus-quickstart-native"
spec:
  resources:
    limits:
      cpu: '4'
      memory: 4Gi

```

The following `oc patch` command adds these `limits`, and `oc start-build` launches a new build:

```

oc patch bc/quarkus-quickstart-native -p
'{"spec":{"resources":{"limits":{"cpu":"4", "memory":"4Gi}}}}'

oc start-build quarkus-quickstart-native

```

# Building a minimal runtime container

As an alternative to creating an application from the S2I build process one can use chained builds to produce a runner image that is minimal in size and does not contain all the dependencies required to build the application.

The following command will create a chained build that is triggered whenever the quarkus-quickstart-native is built.

```
oc new-build --name=minimal-quarkus-quickstart-native \
  --docker-image=registry.access.redhat.com/ubi7-dev-preview/ubi-
minimal \
  --source-image=quarkus-quickstart-native \
  --source-image-path='/home/quarkus/application:.' \
  --dockerfile=$'FROM registry.access.redhat.com/ubi7-dev-
preview/ubi-minimal:latest\nCOPY application /application\nCMD
/application\nEXPOSE 8080'
```

To create a service from the minimal build run the following command:

```
oc new-app minimal-quarkus-quickstart-native
oc expose svc minimal-quarkus-quickstart-native
```

The end result is an image that is less than 40 MB in size (compressed) and does not contain build dependencies like GraalVM, OpenJDK, Maven, etc.

## *Creating build without config.*

The minimal build is depending on the S2I build since it is using the output (native runnable application) from the S2I build. However, you do not need to create an application with `oc new-app`. Instead you could use `oc new-build` like this:



```
oc new-build quay.io/quarkus/ubi-quarkus-native-
s2i:19.3.1-java8~https://github.com/quarkusio/quarkus-
quickstarts.git \
  --context-dir=getting-started --name=quarkus
-quickstart-native
```

Like in previous commands, use `quay.io/quarkus/ubi-quarkus-native-s2i:19.3.1-java11` for building a JDK 11 compatible image.

# Deploying the application as Java application in OpenShift

In this section, we are going to leverage the S2I build mechanism of OpenShift. We use a Java S2I Builder, and therefore do not need a `Dockerfile` in this approach. You do not need to locally clone the Git repository, as it will be directly built inside OpenShift. We are going to create an OpenShift `build` executing it:

```
# To build the image on OpenShift
oc new-app registry.access.redhat.com/redhat-openjdk-18/openjdk18-
openshift~https://github.com/quarkusio/quarkus-quickstarts.git
--context-dir=getting-started --name=quarkus-quickstart
oc logs -f bc/quarkus-quickstart

# To create the route
oc expose svc/quarkus-quickstart

# Get the route URL
export URL="http://$(oc get route | grep quarkus-quickstart | awk
'{print $2}')"
curl $URL/hello/greeting/quarkus
```

Your application is accessible at the printed URL.

The `.s2i/environment` file in the quickstart sets required variables for the S2I Builder image to find the Quarkus `runner` JAR, and copy the JARs from the `lib/` directory:

```
MAVEN_S2I_ARTIFACT_DIRS=target
S2I_SOURCE_DEPLOYMENTS_FILTER=*-runner.jar lib
JAVA_OPTIONS=-Dquarkus.http.host=0.0.0.0
AB_JOLOKIA_OFF=true
```

## Going further

This guide covered the deployment of a Quarkus application on OpenShift using S2I. However, there is much more, and the integration with these environments has been tailored to make Quarkus applications execution very smooth. For instance, the health extension can be used for health check; the configuration support allows mounting the application configuration using config map, the metric extension produces data *scrapable* by Prometheus and so on.