

# Quarkus - Generating Kubernetes resources

This guide covers generating Kubernetes resources based on sane defaults and user supplied configuration.

## Prerequisites

To complete this guide, you need:

- roughly 10 minutes
- an IDE
- JDK 1.8+ installed with `JAVA_HOME` configured appropriately
- Apache Maven 3.5.3+
- access to a Kubernetes or cluster (Minikube is a viable options)

## Creating the Maven project

First, we need a new project that contains the Kubernetes extension. This can be done using the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.3.0.Alpha1:create \
  -DgroupId=org.acme \
  -DartifactId=kubernetes-quickstart \
  -DclassName="org.acme.rest.GreetingResource" \
  -Dpath="/greeting" \
  -Dextensions="kubernetes"
cd kubernetes-quickstart
```

## Enable Kubernetes support

Quarkus offers the ability to automatically generate Kubernetes resources based on sane defaults and user supplied configuration. The implementation that takes care of generating the actual Kubernetes resources is provided by [dekorate](#). Currently it supports the generation of resources for vanilla Kubernetes and OpenShift.

When we added the `kubernetes` extension to the command line invocation above, the following dependency was added to the `pom.xml`

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-kubernetes</artifactId>
</dependency>
```

By adding this dependency, we now have the ability to configure the Kubernetes resource generation and application using the usual `application.properties` approach that Quarkus provides. The configuration items that are available can be found in: `io.quarkus.kubernetes.deployment.KubernetesConfig` class. Furthermore, the items provided by `io.quarkus.deployment.ApplicationConfig` affect the Kubernetes resources.

By using the following configuration for example:

```
kubernetes.group=yourDockerUsername # this is optional and defaults
to your username if not set.
quarkus.application.name=test-quarkus-app # this is also optional
and defaults to the project name if not set
```

and following the execution of `./mvnw package` you will notice amongst the other files that are created, two files named `kubernetes.json` and `kubernetes.yml` in the `target/kubernetes/` directory.

If you look at either file you will see that it contains both a Kubernetes `Deployment` and a `Service`.

The full source of the `kubernetes.json` file looks something like this:

```
{
  "apiVersion" : "v1",
  "kind" : "List",
  "items" : [ {
    "apiVersion" : "apps/v1",
    "kind" : "Deployment",
    "metadata" : {
      "labels" : {
        "app" : "test-quarkus-app",
        "version" : "1.0-SNAPSHOT",
        "group" : "yourDockerUsername"
      },
      "name" : "test-quarkus-app"
    },
    "spec" : {
      "replicas" : 1,
      "selector" : {
        "matchLabels" : {
          "app" : "test-quarkus-app",
          "version" : "1.0-SNAPSHOT",
```

```

        "group" : "yourDockerUsername"
    }
},
"template" : {
    "metadata" : {
        "labels" : {
            "app" : "test-quarkus-app",
            "version" : "1.0-SNAPSHOT",
            "group" : "yourDockerUsername"
        }
    },
    "spec" : {
        "containers" : [ {
            "env" : [ {
                "name" : "KUBERNETES_NAMESPACE",
                "valueFrom" : {
                    "fieldRef" : {
                        "fieldPath" : "metadata.namespace"
                    }
                }
            }
        ],
        "image" : "yourDockerUsername/test-quarkus-app:1.0-SNAPSHOT",
        "imagePullPolicy" : "IfNotPresent",
        "name" : "test-quarkus-app"
    } ]
    }
}
}
} ]
}
}

```

An important thing to note about the **Deployment** is that it uses `yourDockerUsername/test-quarkus-app:1.0-SNAPSHOT` as the Docker image of the **Pod**.

Also the **Service** is configured to use container port `8080` (which is automatically picked up by the standard Quarkus configuration).

## Add readiness and liveness probes

By default the Kubernetes resources do not contain readiness and liveness probes in the generated **Deployment**. Adding them however is just a matter of adding the Smallrye Health extension like so:

```

<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-smallrye-health</artifactId>
</dependency>

```

The values of the generated probes will be determined by the configured health properties: `quarkus.smallrye-health.root-path`, `quarkus.smallrye-health.liveness-path` and `quarkus.smallrye-health.readiness-path`. More information about the health extension can be found in the relevant [guide](#).

## Using the Kubernetes client

Applications that are deployed to Kubernetes and need to access the API server, will usually make use of the `kubernetes-client` extension:

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-kubernetes-client</artifactId>
</dependency>
```

To access the API server from within a Kubernetes cluster, some RBAC related resources are required (e.g. a ServiceAccount, a RoleBinding etc.). So, when the `kubernetes-client` extension is present, the `kubernetes` extension is going to create those resources automatically, so that application will be granted the `view` role. If more roles are required, they will have to be added manually.

## Tuning the generated resources using `application.properties`

The `kubernetes` extension allows tuning the generated manifest, using the `application.properties` file. Here are some examples:

### Changing the number of replicas:

To change the number of replicas from 1 to 3:

```
kubernetes.replicas=3
```

### Defining a docker registry and repository

The docker registry and the user of the docker image can be specified, with the following properties:

```
kubernetes.group=myUser
docker.registry=http://my.docker-registry.net
```

Note: These options used to be `quarkus.kubernetes.docker.registry` and `quarkus.kubernetes.group` respectively.

## Adding labels:

To add a new label to all generated resources, say `foo=bar`:

```
kubernetes.labels[0].key=foo
kubernetes.labels[0].value=bar
```

## Customizing the readiness probe:

To set the initial delay of the probe to 20 seconds and the period to 45:

```
kubernetes.readiness-probe.initial-delay-seconds=20
kubernetes.readiness-probe.period-seconds=45
```

Here you can find a complete reference to all the available configuration options:

### Configuration options

The table below describe all the available configuration options.

Table 1. Kubernetes

Property	Type	Description	Default Value
kubernetes.group	String		
kubernetes.name	String		
kubernetes.version	String		
kubernetes.init-containers	Container[]		
kubernetes.labels	Label[]		
kubernetes.annotations	Annotation[]		
kubernetes.env-vars	Env[]		
kubernetes.working-dir	String		
kubernetes.command	String[]		
kubernetes.arguments	String[]		
kubernetes.replicas	int		1
kubernetes.service-account	String		
kubernetes.host	String		

kubernetes.ports	Port[]		
kubernetes.service-type	ServiceType		ClusterIP
kubernetes.pvc-volumes	PersistentVolumeClaimVolume[]		
kubernetes.secret-volumes	SecretVolume[]		
kubernetes.config-map-volumes	ConfigMapVolume[]		
kubernetes.git-repo-volumes	GitRepoVolume[]		
kubernetes.aws-elastic-block-store-volumes	AwsElasticBlockStoreVolume[]		
kubernetes.azure-disk-volumes	AzureDiskVolume[]		
kubernetes.azure-file-volumes	AzureFileVolume[]		
kubernetes.mounts	Mount[]		
kubernetes.image-pull-policy	ImagePullPolicy		IfNotPresent
kubernetes.image-pull-secrets	String[]		
kubernetes.liveness-probe	Probe		( see Probe )
kubernetes.readiness-probe	Probe		( see Probe )
kubernetes.sidecars	Container[]		
kubernetes.expose	boolean		false
kubernetes.headless	boolean		false
kubernetes.auto-deploy-enabled	boolean		false

Properties that use non standard types, can be referenced by expanding the property. For example to define a **kubernetes-readiness-probe** which is of type **Probe**:

```
kubernetes.readiness-probe.initial-delay-seconds=20
kubernetes.readiness-probe.period-seconds=45
```

In this example `initial-delay` and `period-seconds` are fields of the type `Probe`. Below you will find tables describing all available types.

## Basic Types

Table 2. Annotation

Property	Type	Description	Default Value
key	String		
value	String		

Table 3. Label

Property	Type	Description	Default Value
key	String		
value	String		

Table 4. Env

Property	Type	Description	Default Value
name	String		
value	String		
secret	String		
configmap	String		
field	String		

Table 5. Probe

Property	Type	Description	Default Value
http-action-path	String		
exec-action	String		
tcp-socket-action	String		
initial-delay-seconds	int		0
period-seconds	int		30
timeout-seconds	int		10

Table 6. Port

Property	Type	Description	Default Value
name	String		
container-port	int		

host-port	int		0
path	String		/
protocol	Protocol		TCP

Table 7. Container

Property	Type	Description	Default Value
image	String		
name	String		
env-vars	Env[]		
working-dir	String		
command	String[]		
arguments	String[]		
ports	Port[]		
mounts	Mount[]		
image-pull-policy	ImagePullPolicy		IfNotPresent
liveness-probe	Probe		
readiness-probe	Probe		

## Mounts and Volumes

Table 8. Mount

Property	Type	Description	Default Value
name	String		
path	String		
sub-path	String		
read-only	boolean		false

Table 9. ConfigMapVolume

Property	Type	Description	Default Value
volume-name	String		
config-map-name	String		
default-mode	int		384
optional	boolean		false



Table 10. SecretVolume

Property	Type	Description	Default Value
volume-name	String		
secret-name	String		
default-mode	int		384
optional	boolean		false

Table 11. AzureDiskVolume

Property	Type	Description	Default Value
volume-name	String		
disk-name	String		
disk-uri	String		
kind	String		Managed
caching-mode	String		ReadWrite
fs-type	String		ext4
read-only	boolean		false

Table 12. AwsElasticBlockStoreVolume

Property	Type	Description	Default Value
volume-name	String		
volume-id	String		
partition	int		
fs-type	String		ext4
read-only	boolean		false

Table 13. GitRepoVolume

Property	Type	Description	Default Value
volume-name	String		
repository	String		
directory	String		
revision	String		

Table 14. PersistentVolumeClaimVolume

Property	Type	Description	Default Value
----------	------	-------------	---------------

volume-name	String		
claim-name	String		
read-only	boolean		false

Table 15. AzureFileVolume

Property	Type	Description	Default Value
volume-name	String		
share-name	String		
secret-name	String		
read-only	boolean		false

## Docker

Table 16. Docker

Property	Type	Description	Default Value
docker.docker-file	String		Dockerfile
docker.registry	String		
docker.auto-push-enabled	boolean		false
docker.auto-build-enabled	boolean		false

## OpenShift support

To enable the generation of OpenShift resources, you need to include OpenShift in the target platforms:

```
kubernetes.deployment.target=openshift
```

If you need to generate resources for both platforms (vanilla Kubernetes and OpenShift), then you need to include both (coma separated).

```
kubernetes.deployment.target=kubernetes, openshift
```

The OpenShift resources can be customized in a similar approach with Kubernetes.

Table 17. Openshift

Property	Type	Description	Default Value
openshift.group	String		
openshift.name	String		
openshift.version	String		
openshift.init-containers	Container[]		
openshift.labels	Label[]		
openshift.annotations	Annotation[]		
openshift.env-vars	Env[]		
openshift.working-dir	String		
openshift.command	String[]		
openshift.arguments	String[]		
openshift.replicas	int		1
openshift.service-account	String		
openshift.host	String		
openshift.ports	Port[]		
openshift.service-type	ServiceType		ClusterIP
openshift.pvc-volumes	PersistentVolumeClaim Volume[]		
openshift.secret-volumes	SecretVolume[]		
openshift.config-map-volumes	ConfigMapVolume[]		
openshift.git-repo-volumes	GitRepoVolume[]		
openshift.aws-elastic-block-store-volumes	AwsElasticBlockStoreVolume[]		
openshift.azure-disk-volumes	AzureDiskVolume[]		
openshift.azure-file-volumes	AzureFileVolume[]		
openshift.mounts	Mount[]		

openshift.image-pull-policy	ImagePullPolicy		IfNotPresent
openshift.image-pull-secrets	String[]		
openshift.liveness-probe	Probe		( see Probe )
openshift.readiness-probe	Probe		( see Probe )
openshift.sidecars	Container[]		
openshift.expose	boolean		false
openshift.headless	boolean		false
openshift.auto-deploy-enabled	boolean		false

Table 18. S2i

Property	Type	Description	Default Value
s2i.enabled	boolean		true
s2i.docker-file	String		Dockerfile
s2i.registry	String		
s2i.builder-image	String		fabric8/s2i-java:2.3
s2i.build-env-vars	Env[]		
s2i.auto-push-enabled	boolean		false
s2i.auto-build-enabled	boolean		false
s2i.auto-deploy-enabled	boolean		false

## Knative

To enable the generation of Knative resources, you need to include Knative in the target platforms:

```
kubernetes.deployment.target=knative
```

Following the execution of `./mvnw package` you will notice amongst the other files that are created, two files named `knative.json` and `knative.yml` in the `target/kubernetes/` directory.

If you look at either file you will see that it contains a Knative `Service`.

The full source of the `knative.json` file looks something like this:

```

{
  "apiVersion" : "v1",
  "kind" : "List",
  "items" : [ {
    "apiVersion" : "serving.knative.dev/v1alpha1",
    "kind" : "Service",
    "metadata" : {
      "labels" : {
        "app" : "test-quarkus-app",
        "version" : "0.1-SNAPSHOT",
        "group" : "yourDockerUsername"
      },
      "name" : "knative"
    },
    "spec" : {
      "runLatest" : {
        "configuration" : {
          "revisionTemplate" : {
            "spec" : {
              "container" : {
                "image" : "dev.local/yourDockerUsername/test-
quarkus-app:1.0-SNAPSHOT",
                "imagePullPolicy" : "IfNotPresent"
              }
            }
          }
        }
      }
    }
  } ]
}

```

The generated service can be customized using the following properties:

Table 19. Knative

Property	Type	Description	Default Value
knative.group	String		
knative.name	String		
knative.version	String		
knative.labels	Label[]		
knative.annotations	Annotation[]		
knative.env-vars	Env[]		

knative.working-dir	String		
knative.command	String[]		
knative.arguments	String[]		
knative.service-account	String		
knative.host	String		
knative.ports	Port[]		
knative.service-type	ServiceType		ClusterIP
knative.pvc-volumes	PersistentVolumeClaim Volume[]		
knative.secret-volumes	SecretVolume[]		
knative.config-map- volumes	ConfigMapVolume[]		
knative.git-repo- volumes	GitRepoVolume[]		
knative.aws-elastic- block-store-volumes	AwsElasticBlockStoreV olume[]		
knative.azure-disk- volumes	AzureDiskVolume[]		
knative.azure-file- volumes	AzureFileVolume[]		
knative.mounts	Mount[]		
knative.image-pull- policy	ImagePullPolicy		IfNotPresent
knative.image-pull- secrets	String[]		
knative.liveness-probe	Probe		( see Probe )
knative.readiness-probe	Probe		( see Probe )
knative.sidecars	Container[]		
knative.expose	boolean		false