

Quarkus - Reading properties from Consul

This guide explains how your Quarkus application can read configuration properties at runtime from [Consul](#).



This technology is considered preview.

In *preview*, backward compatibility and presence in the ecosystem is not guaranteed. Specific improvements might require to change configuration or APIs and plans to become *stable* are under way. Feedback is welcome on our [mailing list](#) or as issues in our [GitHub issue tracker](#).

For a full list of possible extension statuses, check our [FAQ entry](#).

Prerequisites

To complete this guide, you need:

- less than 15 minutes
- an IDE
- JDK 1.8+ installed with `JAVA_HOME` configured appropriately
- Apache Maven 3.6.3

Solution

We recommend that you follow the instructions in the next sections and create the application step by step.

Introduction

Consul is a versatile system which among other things, provides a distributed Key-Value store that is used in many architectures as a source of configuration for services. This Key-Value store is what the `quarkus-consul-config` extension interacts with in order to allow Quarkus applications to read runtime configuration properties from Consul.

Starting Consul

There are various ways to start Consul that vary in complexity, but for the purposes of this guide, we elect to start a single Consul server with no persistence via Docker, like so:

```
docker run --rm --name consul -p 8500:8500 -p 8501:8501 consul:1.7
agent -dev -ui -client=0.0.0.0 -bind=0.0.0.0 --https-port=8501
```

Please consult the [documentation](#) to learn more about the various Consul installation options.

Creating the Maven project

First, we need a new project. Create a new project with the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.5.2.Final:create \
  -DprojectId=org.acme \
  -DprojectArtifactId=consul-config-quickstart \
  -DclassName="org.acme.consul.config.GreetingResource" \
  -Dpath="/greeting" \
  -Dextensions="consul-config"
cd consul-config-quickstart
```

This command generates a Maven project with a REST endpoint and imports the `consul-config` extension.

If you already have your Quarkus project configured, you can add the `consul-config` extension to your project by running the following command in your project base directory:

```
./mvnw quarkus:add-extension -Dextensions="consul-config"
```

This will add the following to your `pom.xml`:

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-consul-config</artifactId>
</dependency>
```

GreetingController

The Quarkus Maven plugin automatically generated a `GreetingResource` JAX-RS resource in the `src/main/java/org/acme/consul/config/client/GreetingResource.java` file that looks like:

```

package org.acme.consul.config.client;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class GreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "hello";
    }
}

```

As we want to use configuration properties obtained from the Config Server, we will update the `GreetingResource` to inject the `message` property. The updated code will look like this:

```

package org.acme.consul.config.client;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/hello")
public class GreetingResource {

    @ConfigProperty(name = "message", defaultValue="Hello from default")
    String message;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return message;
    }
}

```

Configuring the application

Quarkus provides various configuration knobs under the `quarkus.consul-config` root. For the

purposes of this guide, our Quarkus application is going to be configured in `application.properties` as follows:

```
# use the same name as the application name that was configured
when standing up the Config Server
quarkus.application.name=consul-test
# enable retrieval of configuration from Consul - this is off by
default
quarkus.consul-config.enabled=true
# this is a key in Consul's KV store that the Quarkus application
will read and try to extract properties from
quarkus.consul-config.properties-value-
keys=config/${quarkus.application.name}
```

Add Configuration to Consul

For the previous application configuration to work, we need to add a `config/consul-test` key under Consul's Key Value store. The value of this key will essentially be a properties "file" containing the application configuration. In this case we want to add the following data to the `config/consul-test` key:

```
greeting.message=Hello from Consul
```

When adding this configuration from the UI, Consul will automatically convert the data into the necessary base64 encoding. If you instead add the configuration via the Consul's [REST API](#), make sure to first encode the previous data into base64.



In this use case we made the value of the key as a properties "file", because we used `quarkus.consul-config.properties-value-keys` in the application. The extension also provides the ability to use the raw values of keys when `quarkus.consul-config.raw-value-keys` is used. Furthermore, these two properties can be used simultaneously, while each one also supports setting multiple keys.

Package and run the application


Run the application with: `./mvnw compile quarkus:dev`. Open your browser to <http://localhost:8080/greeting>.

The result should be: `Hello from Consul` as it is the value obtained from the Consul Key Value store.

Run the application as a native executable

You can of course create a native image using the instructions of the [Building a native executable guide](#).

Configuration Reference

 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
<code>quarkus.consul-config.enabled</code> If set to true, the application will attempt to look up the configuration from Consul	boolean	<code>false</code>
<code>quarkus.consul-config.agent.host-port</code> Consul agent host	host:port	<code>localhost:8500</code>
<code>quarkus.consul-config.agent.use-https</code> Whether or not to use HTTPS when communicating with the agent	boolean	<code>false</code>
<code>quarkus.consul-config.agent.token</code> Consul token to be provided when authentication is enabled	string	
<code>quarkus.consul-config.agent.key-store</code> KeyStore to be used containing the SSL certificate used by Consul agent Can be either a classpath resource or a file system path	path	
<code>quarkus.consul-config.agent.key-store-password</code> Password of KeyStore to be used containing the SSL certificate used by Consul agent	string	
<code>quarkus.consul-config.agent.trust-certs</code> When using HTTPS and no keyStore has been specified, whether or not to trust all certificates	boolean	<code>false</code>

<code>quarkus.consul-config.agent.connection-timeout</code>	Duration ?	10S
The amount of time to wait when initially establishing a connection before giving up and timing out. Specify 0 to wait indefinitely.		
<code>quarkus.consul-config.agent.read-timeout</code>	Duration ?	60S
The amount of time to wait for a read on a socket before an exception is thrown. Specify 0 to wait indefinitely.		
<code>quarkus.consul-config.prefix</code>	string	
Common prefix that all keys share when looking up the keys from Consul. The prefix is not included in the keys of the user configuration		
<code>quarkus.consul-config.raw-value-keys</code>	list of string	
Keys whose value is a raw string. When this is used, the keys that end up in the user configuration are the keys specified here with '/' replaced by '.'		
<code>quarkus.consul-config.properties-value-keys</code>	list of string	
Keys whose value represents a properties file. When this is used, the keys that end up in the user configuration are the keys of the properties file, not these keys		
<code>quarkus.consul-config.fail-on-missing-key</code>	boolean	true
If set to true, the application will not start if any of the configured config sources cannot be located		



About the Duration format

The format for durations uses the standard `java.time.Duration` format. You can learn more about it in the [Duration#parse\(\) javadoc](#).

You can also provide duration values starting with a number. In this case, if the value consists only of a number, the converter treats the value as seconds. Otherwise, **PT** is implicitly prepended to the value to obtain a standard `java.time.Duration` format.