Quarkus - Reading properties from Spring Cloud Config Server

This guide explains how your Quarkus application can read configuration properties at runtime from the Spring Cloud Config Server.

> **ℹ** 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 3.5.3+

Solution

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

Stand up a Config Server

To stand up the Config Server required for this guide, please follow the instructions outlined here. The end result of that process is a running Config Server that will provide the `Hello world` value for a configuration property named `message` when the application querying the server is named `a-bootiful-client`.

Creating the Maven project

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

```
mvn io.quarkus:quarkus-maven-plugin:1.3.0.CR2:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=spring-cloud-config-quickstart \

-DclassName="org.acme.spring.cloud.config.client.GreetingResource"
\
    -Dpath="/greeting" \
    -Dextensions="spring-cloud-config-client"
cd spring-cloud-config-quickstart
```

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

GreetingController

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

```java
package org.acme.spring.spring.cloud.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:

```java
package org.acme.spring.spring.cloud.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 default")
    String message;

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

Configuring the application

Quarkus provides various configuration knobs under the `quarkus.spring-cloud-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=a-bootiful-client
# enable retrieval of configuration from the Config Server - this
is off by default
quarkus.spring-cloud-config.enabled=true
# configure the URL where the Config Server listens to HTTP
requests - this could have been left out since
http://localhost:8888 is the default
quarkus.spring-cloud-config.url=http://localhost:8888
```

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 world` as it is the value obtained from the Spring Cloud Config server.

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.

More Spring guides

Quarkus has more Spring compatibility features. See the following guides for more details:

- Quarkus - Extension for Spring DI
- Quarkus - Extension for Spring Data JPA
- Quarkus - Extension for Spring Security

Spring Cloud Config Client Reference

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

Configuration property	Type	Default
quarkus.spring-cloud-config.enabled If enabled, will try to read the configuration from a Spring Cloud Config Server	boolean	false
quarkus.spring-cloud-config.fail-fast If set to true, the application will not stand up if it cannot obtain configuration from the Config Server	boolean	false
quarkus.spring-cloud-config.url The Base URI where the Spring Cloud Config Server is available	string	http://localhost:8888
quarkus.spring-cloud-config.connection-timeout The amount of time to wait when initially establishing a connection before giving up and timing out. Specify 0 to wait indefinitely.	Duration ❓	10S
quarkus.spring-cloud-config.read-timeout The amount of time to wait for a read on a socket before an exception is thrown. Specify 0 to wait indefinitely.	Duration ❓	60S

`quarkus.spring-cloud-config.username` The username to be used if the Config Server has BASIC Auth enabled	string	
`quarkus.spring-cloud-config.password` The password to be used if the Config Server has BASIC Auth enabled	string	

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.