

**Qualcomm Car-to-Cloud Platform**

**Usage of Config Server**

**Version No.1.0**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** |
| **Name** | Edwin | Sudheer | Sudheer |
| **Role** | PAT | Sr Manager | Sr Manager |
| **Signature** |  |  |  |
| **Date** | April 26, 2021 | April 29, 2021 | April 30, 2021 |

### **Table of Contents**

[**Table of Contents** 2](#_Toc71204968)

[1. Configuring the config client application: 3](#_Toc71204969)

[1.1. The bootstrap.properties file: 3](#_Toc71204970)

[1.2. The application.properties: 3](#_Toc71204971)

[1.3. Injecting configuration values and setting up the @RefreshScope annotation: 4](#_Toc71204972)

[2. Configuring the config server application: 5](#_Toc71204973)

[2.1. The application.properties: 5](#_Toc71204974)

[2.2. The bootstrap.properties: 5](#_Toc71204975)

[3. Encryption 6](#_Toc71204976)

[4. Procedure 7](#_Toc71204977)

# Configuring the config client application:

## The bootstrap.properties file:

1. spring.application.name=<This will identify which property file to fetch from the repository for this particular service>
2. spring.profiles.active=<profile of the configuration file>
3. management.endpoints.web.exposure.include=<to expose endpoints beneath base path like '/actuator' like ‘/refresh’,’/health’, etc>
4. spring.cloud.config.fail-fast=<client application will not run if it is set to true when the configuration server is not responding>
5. spring.cloud.config.uri=<The URI of the configuration server should be given>

**Note:** It is possible to access more than one properties files with different application names.

For example, say the repository contains more than one application like “microservice-one.properties” and “global.properties”. In order to get values from both the files we have to give the property “spring.application.name” as:

**spring.application.name=microservice-one, global**

This will load the values from both the property files.

## The application.properties:

1. server.port=<specify the port no to run the application, if needed>

## Injecting configuration values and setting up the @RefreshScope annotation:

The values in the configuration file from the config server can be injected into fields in Spring-managed beans using ‘@Value’ or ‘@ConfigurationProperties’ annotations.

For example,

**@Value(“${sample.config.name:default value}")**

**private String name;**

This will get the value of the property “sample.config.name” from the configuration file and inject into the field ‘name’.

The default values should be specified, else it will give an error on startup.

In order to refresh the properties in real-time using the actuator endpoint, the annotation ‘@RefreshScope’ should be given for the POJO class which is used to get the property values.

**Note:** Add “spring-cloud-starter-config” dependency in the pom.xml instead of “spring-cloud-config-client”.

# Configuring the config server application:

## The application.properties:

1. spring.cloud.config.server.git.uri=<git repository location containing the configuration file(local or cloud)>
2. spring.cloud.config.server.git.username=< access-key of the IAM user to access the AWS Code Commit repository>
3. spring.cloud.config.server.git.password=< secret-key of the IAM user to access the AWS Code Commit repository>
4. spring.cloud.config.label=<specify the label, i.e., branch of the git repository. If not specified, it will look for master branch>

## The bootstrap.properties:

1. encrypt.key=<The encryption key should be provided. This is used for the symmetric encryption>

**Note:**

* + 1. If AWS code commit repository is given, make sure the server spring application has the dependency “aws-java-sdk-codecommit”.
    2. The access key and secret key of an IAM user who have access to the AWS code commit repository should be specified in the git username and password properties for connection.
    3. The property files are stored in the repository in the format :

**{application}-{profile}.properties**

**Example:** microservice-development.properties

Where {application} placeholder refers to the client’s application name and {profile} refers to the different environments like development, testing, etc.

# Encryption

The encryption key given in the bootstrap.properties is used for encryption. A string can be encrypted by using the given endpoint:

http://<ip of server config>:<port no>/encrypt

**Example: http://localhost:8090/encrypt**

Enter the string to be encrypted in the body column and send a POST request. The response will be an encrypted value. It can be used in the configuration files in order to store sensitive values securely.“{cipher}” keyword is to be used in front of the encrypted value stored in the configuration file.

**Example: config.application.password={cipher}Qweyuy@#3+dnjbxIRUFJDCMKNVOAv**

The server reads the value, decrypts it using the encryption key and the value will be given to the client in readable form.

The client application is not involved in the encryption process.

# Procedure

After configuring both client config and server config application, run the server and the client applications. The client has received the property values from the configuration file in the git repository via the server application which is tightly coupled with the git repository as soon as both start running.

Now, if a change is made in the configuration file, the server will get the new values in real-time.

We can retrieve the configuration of the config file using the endpoint:

http://<ip of server config>:<port no>/<configuration file name>/<profile if any, else default>/<label if any, else master>

**Example: http://localhost:8090/config-client/development/main**

(Here, ‘label’ refers to the branch of the repository and ‘profile’ refers to different environments like development, testing, etc.)

In order to get the configuration values in the client side a ‘/refresh’ endpoint is used. For that a POST request is send to the following URL:

http://<ip of client config>:<port no>/actuator/refresh

**Example: http://localhost:8000/actuator/refresh**

Now the updated configuration values are retrieved by client application from the server config application.