

**Qualcomm Car-to-Cloud Platform**

**Usage of Config Server**

**Version No.2.0**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** |
| **Name** | Edwin | Sudheer | Sudheer |
| **Role** | PAT | Sr Manager | Sr Manager |
| **Signature** |  |  |  |
| **Date** | April 26, 2021 | June 08, 2021 | June 08, 2021 |

### **Table of Contents**

[**Table of Contents** 2](#_Toc74086328)

[1. Configuring the config client application: 3](#_Toc74086329)

[1.1. The bootstrap.properties file: 3](#_Toc74086330)

[1.2. The application.properties file: 4](#_Toc74086331)

[1.3. Configuring Environment Variables 4](#_Toc74086332)

[1.4. Placing a configuration file in the S3 backend 5](#_Toc74086333)

[1.5. Injecting configuration values and setting up the @RefreshScope annotation: 6](#_Toc74086334)

[2. Configuring the config server application: 7](#_Toc74086335)

[2.1. The application.properties: 7](#_Toc74086336)

[2.2. The bootstrap.properties: 7](#_Toc74086337)

[2.3. Configuring Environment Variables 7](#_Toc74086338)

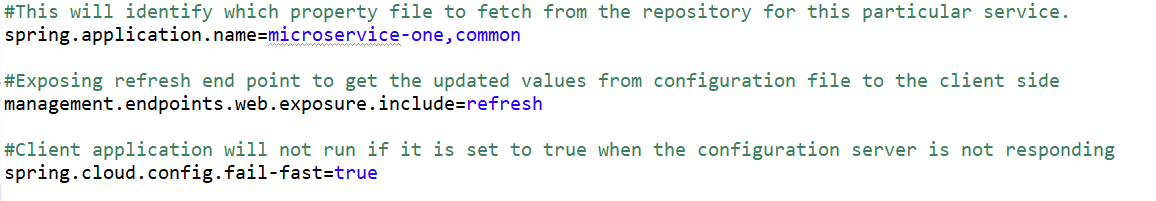
[3. Encryption 8](#_Toc74086339)

[4. Procedure 9](#_Toc74086340)

# 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. management.endpoints.web.exposure.include=<to expose endpoints beneath base path like '/actuator', ‘/refresh’, ’/health’ etc>
3. spring.cloud.config.fail-fast=<client application will not run if it is set to true when the configuration server is not responding>



**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 “common.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, common**

This will load the values from both the property files.

## The application.properties file:

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

## Configuring Environment Variables

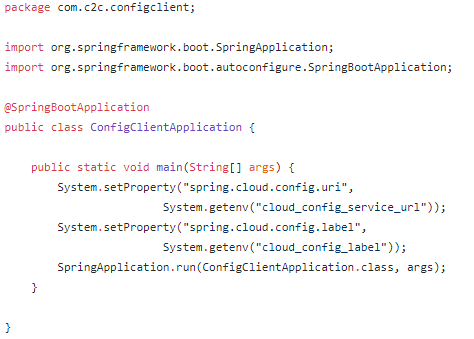
The environment variables in the system properties have to be configured for getting the server URL and the environment value (development, testing, etc.)

**Example:**

|  |  |
| --- | --- |
| **Variable** | **Value** |
| cloud\_config\_label | development |
| cloud\_config\_service\_url | http://localhost:8888 |

Here, “cloud\_config\_label” indicates the environment to be specified and “cloud\_config\_service\_url” indicates the URL of Config server.

Also, make sure the client application properly accesses these values using ***System.getenv()*** method and sets it as the system property using ***System.setProperty()***as given below.



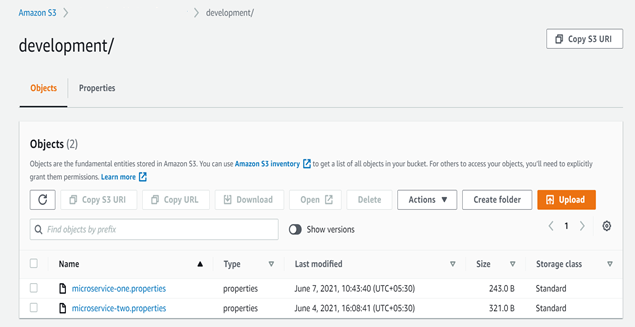
## Placing a configuration file in the S3 backend

* + - 1. The configuration files are placed in the S3 bucket “[**c2c-\*\*\*\*\*\*\*\*\*-gbl-s3-config-server**](https://s3.console.aws.amazon.com/s3/buckets/c2c-eus-dev-gbl-s3-config-server)”
      2. Create folders inside the bucket corresponding to the environment like “**development**”, “**testing**”, etc.
      3. Place the configuration file inside the required folder with the naming convention as:

**{application}.properties**

Where {application} placeholder should be replaced with the client’s application name

**Example:** microservice-one.properties



## 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.profiles.active=awss3

<This is used to specify the profile active in order to use it as the backend repo for server. In our case, we are using AWS S3 as the backend>

* 1. server.port=<specify the port no, if required>

## The bootstrap.properties:

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

**Note:**

If AWS S3 is given as the backend, make sure the server spring application has the dependency “**aws-java-sdk-s3**”.

## Configuring Environment Variables

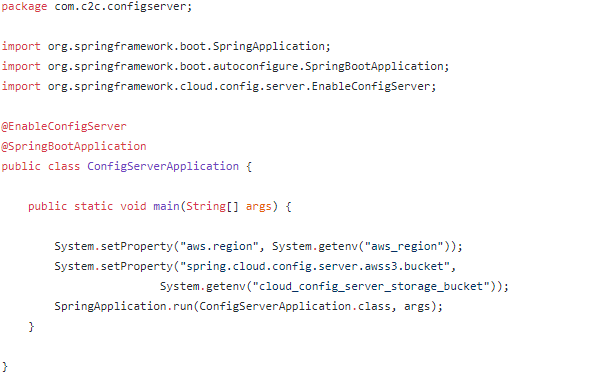
The environment variables in the system properties have to be configured for getting the bucket name in S3 and the region corresponding to that bucket.

Example:

|  |  |
| --- | --- |
| **Variable** | **Value** |
| aws\_region | us-east-1 |
| cloud\_config\_server\_storage\_bucket | c2c-eus-dev-gbl-s3-config-server |

Here, “aws\_region” indicates the region of the bucket and “cloud\_config\_server\_storage\_bucket” indicates the bucket name.

Also, make sure the application properly access these values using ***System.getenv()*** method and sets it as the system property using ***System.setProperty()*** as given below.



# 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:8888/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}QweyuyUFJDCMKNVOAv**

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

**Note:** The client application is not involved in the encryption process.

# Procedure

The configuration of both config client and config server has to be done properly. ***For a normal user configuring the client application, please refer to sections 1 and 3***.

Once the server and client start running, the client receives the property values from the configuration files in the s3 backend via the server application.

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:8888/** **microservice-one/default/development**

(Here, ‘profile’ is set as default and ‘label’ refers to the environment of the application 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 config server service.