**Spring Cloud Config Server**

Spring Cloud Config provides server and client-side support for externalized configuration in a distributed system. With the Config Server you have a central place to manage external properties for applications across all environments.

As an application moves through the deployment pipeline from dev to test and into production you can manage the configuration between those environments and be certain that applications have everything they need to run when they migrate.

In addition, we can change properties without restarting application by using Spring Actuator ‘/refresh’ endpoint.

**Spring Cloud Config Server(Spring Boot App)**

Create config server using spring boot app add following POM and Application Properties

POM Details

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

Application Properties

spring:

application:

name: spring-config-server

cloud:

config:

server:

git:

uri: <https://github.com/config-server/config-prop> (Remote GIT Repo)

force-pull: **true**

clone-on-start: **true**

server:

port: 8888

management:

security:

enabled: **false**

In above prop we have given application name, GIT URI to which server should point, Server Port and for testing purpose we are disabling spring actuator end point security in production we need to enable spring security

Here GIT URI Can be used in two ways in above prop file we are using remote GIT URI. How to use local repo explained below

**Creation of local Repo and using it in GIT URI link**

Create repo and add properties file (**Here most important thing is properties file name should be same as application name(client) this is how config server will identify it’s respective client**)

Eg: spring-config-client.properties

Later install Git an open GIT bash in local repository and perform following commands

git init .

git add -A .

git commit -m "Add application.properties"

In Above step you are committing files is repo locally.

Then give this local repo path in GIT URI in application.yml

spring:

application:

name: spring-config-server

cloud:

config:

server:

git:

uri: <file:///c:/repo-name>

file:/// must be appended to URI if you are working on windows else, direct path can be given.

**GIT REMOTE URL**

spring:

application:

name: spring-config-server

cloud:

config:

server:

git:

uri: <https://github.com/config-server/config-prop>

If you don’t use HTTPS and user credentials, SSH should also work out of the box when you store keys in the default directories (~/.ssh) and the uri points to an SSH location, e.g. "[git@github.com](mailto:git@github.com):configuration/cloud-configuration". It is important that an entry for the Git server be present in the ~/.ssh/known\_hosts file and that it is in ssh-rsa format. Other formats (like ecdsa-sha2-nistp256) are not supported. To avoid surprises, you should ensure that only one entry is present in the known\_hosts file for the Git server and that it is matching with the URL you provided to the config server. If you used a hostname in the URL, you want to have exactly that in the known\_hosts file, not the IP. The repository is accessed using JGit, so any documentation you find on that should be applicable. HTTPS proxy settings can be set in ~/.git/config or in the same way as for any other JVM process via system properties (-Dhttps.proxyHost and -Dhttps.proxyPort).

**If you do not know, where your ~/.git directory is use git config --global to manipulate the settings (e.g. git config --global http.sslVerify false)**

**Spring-Config-Client(Spring Boot APP)**

**POM.xml**

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-config</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

**Application.yml**

spring:

application:

name: spring-config-client

cloud:

config:

uri: http://localhost:8888

profiles:

active: development

server:

port: 8001

management:

security:

enabled: **false**

**Bootstrap.yml(should be in resources)(this one was working while trying on Docker)**

spring:

application:

name: spring-config-client

profiles:

active: development

cloud:

config:

uri: http://spring-config-server:8000

enabled: **true**

server:

port: 8001

management:

security:

enabled: **false**

In Cloud config URI we are point towards Config Server where all configuration details will be given. And hosting it on Port 8001

Now we’re able to start our server. The *Git*-backed configuration API provided by our server can be queried using the following paths:

/{application}/{profile}[/{label}]

/{application}-{profile}.yml

/{label}/{application}-{profile}.yml

/{application}-{profile}.properties

/{label}/{application}-{profile}.properties

In above properties file we are giving **profiles-active** as **development** so that API will look for Configuration file with applicationname-profile.properties

In our case **spring-config-client-development.properties** willbe picked.