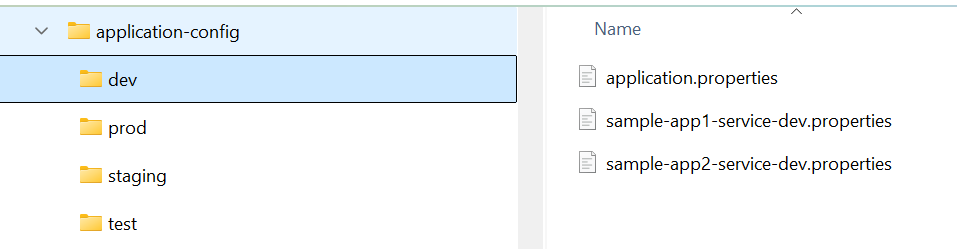
**Spring Cloud Config Server and Client – 2024**

**Directory Structure of Configuration Files: The following is the directory structure of the application configuration.**



**Spring Cloud Config Server**

You can run Spring cloud config server in two modes with connection to Gitlab/Github and without git connecting to local directory structure. When you run Cloud Config Server without git connection, you have to run as a **native** profile.

**How to create Cloud Config Server**

**Pom.xml Relevant Portion**

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.2.4</version>  
 <relativePath />  
</parent>

<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-config-server</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
 </dependencies>  
<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version> 🡸 2023.0.0  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
</dependencyManagement>

**Create main application as below**

**@EnableConfigServer  
@SpringBootApplication(exclude = SecurityAutoConfiguration.class)**  
public class SampleCloudConfigServerApp {  
 public static void main(String[] args) {  
 SpringApplication.*run*(SampleCloudConfigServerApp.class, args);  
 }  
}

**Create the following properties files**

**application-dev.properties** 🡸 To run with git configuration

**application-native.properties** 🡸 To run with native profile connecting to local directory structure

**application-native.properties**

*# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Spring Core Properties \*\*\*\*\*\*\*\*\*\*\*\*\*\**server.port=8888 *# --------------------------------------------------------*

*# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Spring cloud Properties \*\*\*\*\*\*\*\*\*\*\*\*\*\****spring.cloud.config.server.git.default-label=native**  
*# Change the location of c3-config as per your location***local.directory=E:/sample-cloud-config-server-without-git-2024/application-config**  
*#search.profile.directory=file:///${local.directory}/dev***search.profile.directory=file:///${local.directory}/staging** 🡸 Where to search  
spring.cloud.config.server.native.search-locations=${search.profile.directory}  
  
**application-dev.properties**

*# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Spring Core Properties \*\*\*\*\*\*\*\*\*\*\*\*\*\**spring.application.name=configserver  
server.port=8888  
*# --------------------------------------------------------  
  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Spring cloud Properties \*\*\*\*\*\*\*\*\*\*\*\*\*\*  
# For Git URL***spring.cloud.config.server.git.username=ddlab***# gitlab password is the token***spring.cloud.config.server.git.password=glpat-jPyRgdchXuqcMd4jCuDK  
spring.cloud.config.server.git.default-label=main 🡸 Branch name of the config repository  
spring.cloud.config.server.git.uri=https://gitlab.com/ddlab/application-config.git  
spring.cloud.config.server.git.clone-on-start=true***#spring.cloud.config.server.git.basedir=file://${user.dir}/cloned\_configurations*spring.cloud.config.server.git.skip-ssl-validation=true  
spring.cloud.config.server.git.skipSslValidation=true  
  
*# Search Path*spring.cloud.config.server.git.searchPaths=staging,dev,test 🡸 Search Path  
*# --------------------------------------------------------*

To Start Cloud Config Server in native profile mode, Run with the following VM argements

**--add-opens java.base/java.lang=ALL-UNNAMED -Dspring.profiles.active=native**

To Start Cloud Config Server in dev profile mode to connect to Gitlab, Run with the following VM argements

**--add-opens java.base/java.lang=ALL-UNNAMED -Dspring.profiles.active=dev**

**Spring Cloud Config Client**

Spring Cloud Config Client is a general/any microservice which will connect to Spring Cloud Config Server with host and port. Create a simple microservice and define the following properties based on the environments.

Pom.xml Relevant portion

<parent>

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

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

<version>3.2.4</version>

<relativePath />

</parent>

<properties>

<java.version>17</java.version>

<spring-cloud.version>2023.0.0</spring-cloud.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version> 🡸 2023.0.0

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**application-dev.properties**

# you can also mention the port and context path in the config properties file for cloud config

server.port=8080

server.servlet.contextPath=/sample-app1-service

# Mention the name of the microservice below

spring.application.name=sample-app1-service-dev

spring.config.import=optional:configserver:http://localhost:8888

**application-staging.properties**

# you can also mention the port and context path in the config properties file for cloud config

server.port=8080

server.servlet.contextPath=/sample-app1-service

# Mention the name of the microservice below

spring.application.name=sample-app1-service

spring.config.import=optional:configserver:http://localhost:8888

Note: **It is not important whether you run Cloud Config Server in native mode or dev/git mode, you have to run the client applications or microservice based on the environments**.

Sample Rest Controller is given below.

@RestController

**public** **class** SampleAppController {

@Value("${app.message: DefaultMessage}")

**private** String message;

@Value("${common.config.name: NoConfigInfo}")

**private** String commonConfigName;

@Value("${key.config.name: NoKeyConfig}")

**private** String envKey;

@GetMapping(path = "/v1/info")

**public** ResponseEntity<InfoVO> getMessage() {

InfoVO infoVo = **new** InfoVO(message, commonConfigName, envKey);

**return** **new** ResponseEntity<>(infoVo, HttpStatus.***OK***);

}

}

Access the application as GET <http://localhost:8080/sample-app1-service/v1/info>