```
@EnableConfigServer
@SpringBootApplication
@EnableCaching
@Slf4j
public class ConfigServerApplication extends
SpringBootServletInitializer {

    private static Class<ConfigServerApplication>
applicationClass = ConfigServerApplication.class;
    private static String propertySourceName;
    private static String revision;

    @Autowired
    Environment env;

    @Autowired
    Monitoring monitor;

    public static void main(String[] args) {
        initConfiguration(new
SpringApplicationBuilder()).build().run();
    }

    private static SpringApplicationBuilder
initConfiguration(SpringApplicationBuilder applicationBuilder) {
        try {
            Properties prop = getConfigProperties();

applicationBuilder.application().setDefaultProperties(prop);
        } catch (Exception ex) {

log.error(LoggerUtil.fmt(ConfigConstants.MSG_ID_50001,
ConfigConstants.MSG_NAME_50001,
                    ConfigConstants.MSG_DETAIL_50001), ex);
        }
        return applicationBuilder.sources(applicationClass);
    }

    /*
     * Retrieve config repo json from cloudant and add to
properties object
     */
    private static Properties getConfigProperties() {
        Properties prop = new Properties();
        try {
            JSONObject configObj =
CloudantClientManager.getConfigRepos();
```

```
            if (null == configObj) {

log.info(LoggerUtil.fmt(ConfigConstants.MSG_ID_30002,
ConfigConstants.MSG_NAME_30002,
ConfigConstants.MSG_DETAIL_30002));
                return null;
            }
            Iterator<?> keys = configObj.keys();
            while (keys.hasNext()) {
                String key = (String) keys.next();
                String value = configObj.getString(key);
                //log.info(key + " : " + value);
                if (key.startsWith("common")) //Application
related properties
                    System.setProperty(key, value);
                else {
                    if
(key.equals("spring.cloud.config.server.git.repos")) {
                        String repos =
configObj.get("spring.cloud.config.server.git.repos").toString()
;
                        JSONArray reposArray = new
JSONArray(repos);
                        for (int i = 0; i < reposArray.length();
i++) {
                            JSONObject eachRepo =
reposArray.getJSONObject(i);
                            JSONObject connectionObj =
eachRepo.getJSONObject("connection");
                            Iterator<?> subKeys =
connectionObj.keys();
                            while (subKeys.hasNext()) {
                                String subKey = (String)
subKeys.next();
                                String subValue =
connectionObj.getString(subKey);

prop.setProperty("spring.cloud.config.server.git.repos."
                                                +
eachRepo.getString("name") + "." + subKey, subValue);
                            }
                        }
                    } else
                        prop.setProperty(key, value);
                }
            }
        } catch (Exception ex) {
```

```
log.error(LoggerUtil.fmt(ConfigConstants.MSG_ID_50001,
ConfigConstants.MSG_NAME_50001,
                    ConfigConstants.MSG_DETAIL_50001), ex);
        }
        return prop;
    } //getConfigProperties end

    /**
     * SpringApplicationBuilder
configure(SpringApplicationBuilder application) will trigger to
initialize the application settings
     */
    @Override
    protected SpringApplicationBuilder
configure(SpringApplicationBuilder applicationBuilder) {
        return initConfiguration(applicationBuilder);
    }

    @Scheduled(fixedRate =
ConfigConstants.reloadConfigSchedularFrequency, initialDelay =
ConfigConstants.reloadConfigSchedularFrequency)
    public String reloadConfigRepo() {
        log.info(LoggerUtil.fmt(ConfigConstants.MSG_ID_30001,
ConfigConstants.MSG_NAME_30001,
ConfigConstants.MSG_DETAIL_30001_1));
        String response = null;
        try {
            if ((null == propertySourceName || null == revision)
&& env instanceof ConfigurableEnvironment) {
                //Getting revision number very first time after
server boot
                for (PropertySource<?> propertySource :
((ConfigurableEnvironment) env).getPropertySources()) {
                    if (propertySource instanceof
EnumerablePropertySource
                            &&
propertySource.getProperty(ConfigConstants.configRepoIdentifier)
!= null) {
                        propertySourceName =
propertySource.getName();
                        revision = (String)
propertySource.getProperty(ConfigConstants.configRepoIdentifier)
;
                    }
                }
            }
```

```
            Properties prop = getConfigProperties();
            if (null != revision && null !=
prop.getProperty(ConfigConstants.configRepoIdentifier)
                    && !
revision.equals(prop.getProperty(ConfigConstants.configRepoIdent
ifier))) {

log.info(LoggerUtil.fmt(ConfigConstants.MSG_ID_30001,
ConfigConstants.MSG_NAME_30001,
ConfigConstants.MSG_DETAIL_30001_2));
                revision =
prop.getProperty(ConfigConstants.configRepoIdentifier);
                MutablePropertySources sources =
((ConfigurableEnvironment) env).getPropertySources();
                sources.replace(propertySourceName, new
PropertiesPropertySource(propertySourceName, prop));
                response = "Reloaded config successfully";

                //can add refresh call from here to load new
repo right away to spring context
                monitor.refreshAllRepos();

            } else
                response = "Reload ignored as no changes
identified in config";

        } catch (Exception ex) {

log.error(LoggerUtil.fmt(ConfigConstants.MSG_ID_50026,
ConfigConstants.MSG_NAME_50026,
                    ConfigConstants.MSG_DETAIL_50026), ex);
        }
        return response;
    } //reloadConfigRepo end

}
```