**Centralized Configuration Pattern**

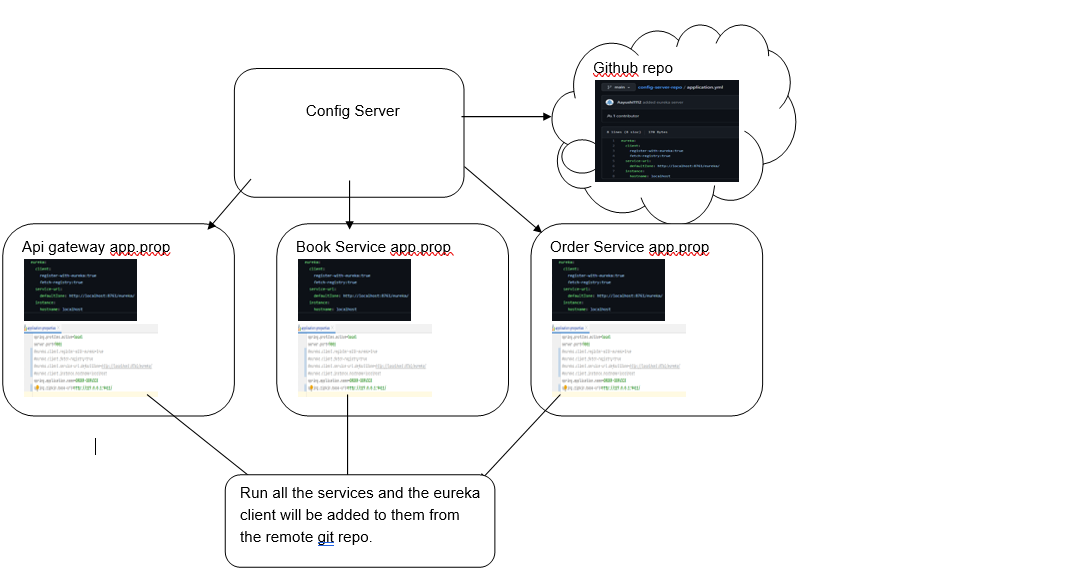
If we have 100-1000 microservices and they all have some common configuration, so its always good to externalize the common configuration of each microservice and maintain it in one common place.

This helps in reducing rework and code repetition and increases reuasbility and maintainability.

We have been repeating configuration for Eureka server(Discovery service) in all the client application, so this is one of the good example to externalize.

1. Create a new github or bitbucket or any git server repo and bring the configuration there
2. Create config server(microservice) and connect to github
3. Configure all the microservices to connect to config server and read configuration

**Architecture**

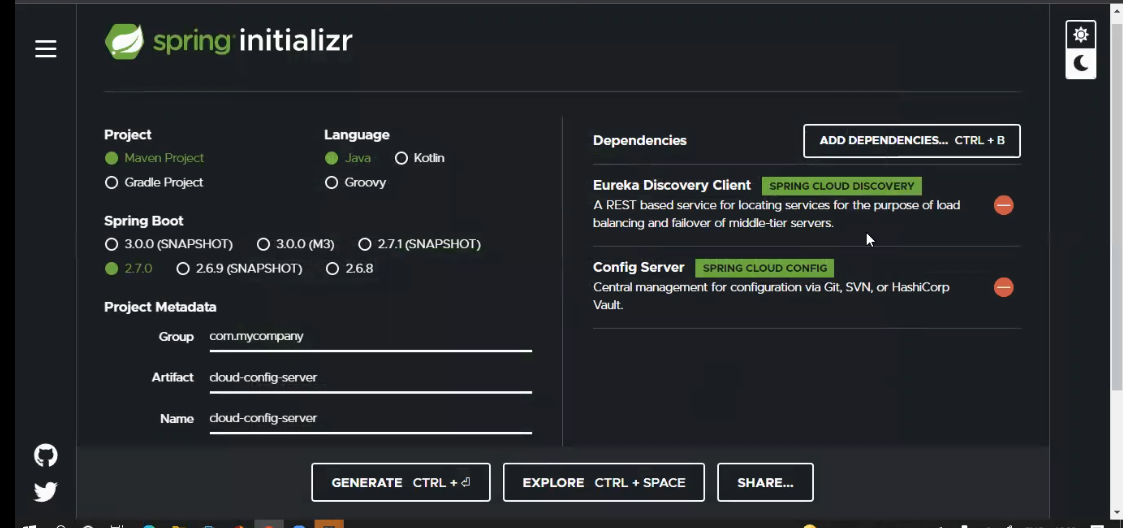
****

**A diagram of a service

Description automatically generated**

Steps:

* Create new springboot project as config-server



* Add Eureka client and Config server dependency
* Download the project and import to intellij
* In main class @EnableEurekaClient & @EnableConfigServer



* Create application.yml file and Add below configuration

|  |
| --- |
| spring.application.name=CONFIG-SERVER  server.port=8085  spring.cloud.config.server.git.uri=https://gitlab.com/aayushi2853213/microservice-configurations  spring.cloud.config.server.git.username=Aayushi1995  spring.cloud.config.server.git.password=Aayusal1995\*  spring.cloud.config.server.git.clone-on-start=true |

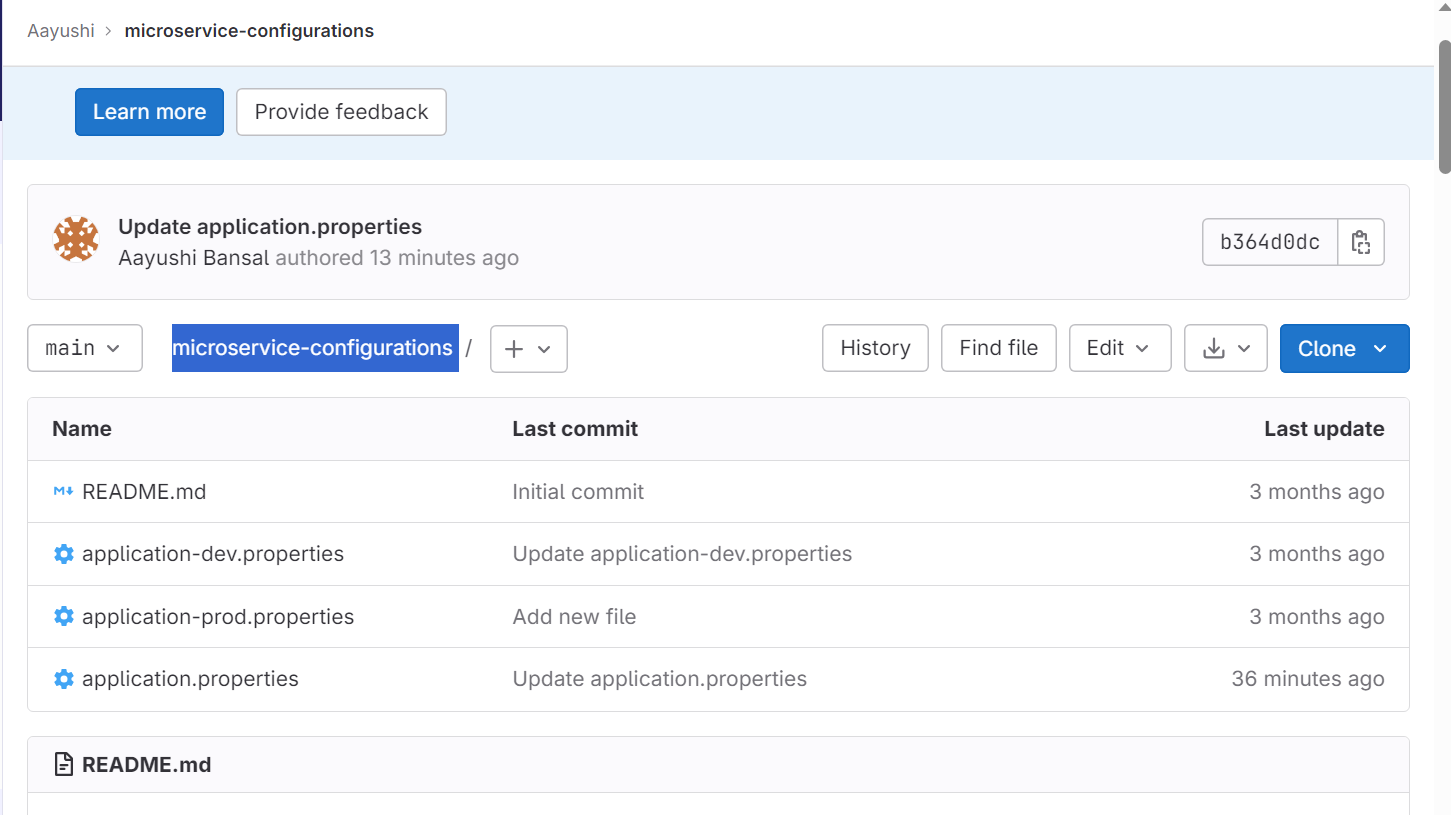
A computer screen shot of a computer code

Description automatically generated

**CREATING GITHUB REPO**

* Create new github repo with name config-server-repo
* Create application.properties in github file and move the eureka client config from all places to here and remove them from all the places (cut paste)

|  |
| --- |
|  |

A screenshot of a computer

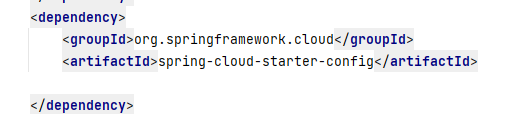
Description automatically generated

**STEPS IN MICROSERVICES WHO WANT TO CONNECT TO CONFIG SERVER APPLICATION**

* Follow below steps in all the microservices so that they can now connect to config server to read the common configuration

In microservice add below dependency(in api gateway,book service,order service etc so that they all can register themselves with config server)

|  |
| --- |
| 1. <dependency> 2. <groupId>org.springframework.cloud</groupId> 3. <artifactId>spring-cloud-starter-config</artifactId> 4. </dependency> 5. <dependency> 6. <groupId>org.springframework.cloud</groupId> 7. <artifactId>spring-cloud-starter-bootstrap</artifactId> 8. </dependency> |



b.) In the microservice application.propertiesfile add configuration so that it can connect to the config-server and fetch the configuration.

|  |
| --- |
| #config server  spring.config.import=configserver:http://localhost:8085  spring.config.cloud.uri=http://localhost:8085  #spring.cloud.config.enabled=false  spring.cloud.config.profile=default |

A blue and white screen

Description automatically generated

Application.proepties file is used by springboot to initialize spring context when the application starts but we need some cloud configuration to be bootstrapped(ready) before application.yml is read by spring to initialize its context hence for that we have to create a bootstrap.yml file.

**Note:**

* When a springboot application starts it loads the context first of all
* Loading the context means all the data which is in the application.properties file will be loaded and memory will be given to that.
* But now as we have our common configurations in the github repo so we want to load them first.
* Thus we have a bootstrap.yml file which will connect all our components to the config server initially.
* Then config server will fetch dependencies from github repo and put in the application.properties of all applications and then their context will be loaded.
* After that all our files will work and register themselves on the eureka client

TESTING THE CONFIG SERVER

