**Spring Boot Docker Kubernetes – Configuration Management In Microservices**

**Step -1 – Introduction to Configuration Management inside microservices  
  
Step -2 – How Configuration work in Spring Boot  
  
Step -3 – Reading configuration using @Value annotation  
accounts\src\main\resources\application.yml**…

build:

version: "1.0"

package com.eazybytes.accounts.controller;

import lombok.AllArgsConstructor;

…

@AllArgsConstructor

@Validated

public class AccountsController {

private IAccountService iAccountsService;

@Value("${build.version}")

private String buildVersion;

public AccountsController(IAccountService accountService) {

this.iAccountsService = accountService;

}

…

@Operation(

summary = "Get Build information",

description = "Get Build information that is deployed into accounts microservice"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@GetMapping("/build-info")

public ResponseEntity<String> getBuildInfo() {

return ResponseEntity

.status(HttpStatus.OK)

.body(buildVersion);

}

}

GET http://localhost:8080/api/build-info

**Update Loans for @Value annotation  
loans\src\main\resources\application.yml**

…

build:

version: "2.0"  
  
package com.eazybytes.loans.controller;

…  
~~@AllArgsConstructor~~

public class LoansController {

private ILoansService iLoansService;

public LoansController(ILoansService iLoansService) {

this.iLoansService = iLoansService;

}

@Value("${build.version}")

private String buildVersion;

…

@Operation(

summary = "Get Build information",

description = "Get Build information that is deployed into cards microservice"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@GetMapping("/build-info")

public ResponseEntity<String> getBuildInfo() {

return ResponseEntity

.status(HttpStatus.OK)

.body(buildVersion);

}

}

GET http://localhost:8090/api/build-info

**Update Cards for @Value annotation  
cards\src\main\resources\application.yml**

…

build:

version: "3.0"

package com.eazybytes.cards.controller;

~~@AllArgsConstructor~~

public class CardsController {

private ICardsService iCardsService;

public CardsController(ICardsService iCardsService) {

this.iCardsService = iCardsService;

}

@Value("${build.version}")

private String buildVersion;

…

@Operation(

summary = "Get Build information",

description = "Get Build information that is deployed into cards microservice"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@GetMapping("/build-info")

public ResponseEntity<String> getBuildInfo() {

return ResponseEntity

.status(HttpStatus.OK)

.body(buildVersion);

}

}

GET <http://localhost:9000/api/build-info>

**Step -4 – Reading configuration using Environment interface**

package com.eazybytes.accounts.controller;

…

public class AccountsController {

…

@Operation(

summary = "Get Java version",

description = "Get Java versions details that is installed into accounts microservice"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@GetMapping("/java-version")

public ResponseEntity<String> getJavaVersion() {

return ResponseEntity

.status(HttpStatus.OK)

.body(environment.getProperty("JAVA\_HOME"));

}

}

GET <http://localhost:8080/api/java-version>

**Updating Loans and Cards configuration using Environment interface**package com.eazybytes.loans.controller;

…

public class LoansController {

…

@Autowired

private Environment environment;

…

@Operation(

summary = "Get Java version",

description = "Get Java versions details that is installed into accounts microservice"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@GetMapping("/java-version")

public ResponseEntity<String> getJavaVersion() {

return ResponseEntity

.status(HttpStatus.OK)

.body(environment.getProperty("JAVA\_HOME"));

}

}

GET <http://localhost:8090/api/java-version>

package com.eazybytes.cards.controller;

…

public class CardsController {

…

@Autowired

private Environment environment;

…

@Operation(

summary = "Get Java version",

description = "Get Java versions details that is installed into accounts microservice"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@GetMapping("/java-version")

public ResponseEntity<String> getJavaVersion() {

return ResponseEntity

.status(HttpStatus.OK)

.body(environment.getProperty("JAVA\_HOME"));

}

}

GET <http://localhost:9000/api/java-version>

**Step -5 – Reading configuration using @ConfigurationProperties  
accounts\src\main\resources\application.yml**

…

accounts:

message: "Welcome to EazyBank accounts related local APIs "

contactDetails:

name: "John Doe - Developer"

email: "john@eazybank.com"

onCallSupport:

- (555) 555-1234

- (555) 523-1345

package com.eazybytes.accounts.dto;

@ConfigurationProperties(prefix = "accounts")

public record AccountsContactInfoDto(String message, Map<String, String> contactDetails, List<String> onCallSupport) {

}

package com.eazybytes.accounts.controller;

…

public class AccountsController {

…

@Autowired

private AccountsContactInfoDto accountsContactInfoDto;

…

@Operation(

summary = "Get Contact Info",

description = "Contact Info details that can be reached out in case of any issues"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@GetMapping("/contact-info")

public ResponseEntity<AccountsContactInfoDto> getContactInfo() {

return ResponseEntity

.status(HttpStatus.OK)

.body(accountsContactInfoDto);

}

}

package com.eazybytes.accounts;

…

@EnableConfigurationProperties(value = {AccountsContactInfoDto.class})

…

public class AccountsApplication {

…

}

GET<http://localhost:8080/api/contact-info>

**Update Loans for Reading configuration using @ConfigurationProperties**loans\src\main\resources\application.yml  
**…**loans:

message: "Welcome to EazyBank loans related local APIs "

contactDetails:

name: "Amaal Grega - Developer"

email: "amaal@eazybank.com"

onCallSupport:

- (452) 456-2176

- (546) 764-8934

package com.eazybytes.loans.dto;

@ConfigurationProperties(prefix = "loans")

public record LoansContactInfoDto(String message, Map<String, String> contactDetails, List<String> onCallSupport) {

}

package com.eazybytes.loans.controller;

…

public class LoansController {

…

@Autowired

private LoansContactInfoDto loansContactInfoDto;

…

@Operation(

summary = "Get Contact Info",

description = "Contact Info details that can be reached out in case of any issues"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@GetMapping("/contact-info")

public ResponseEntity<LoansContactInfoDto> getContactInfo() {

return ResponseEntity

.status(HttpStatus.OK)

.body(loansContactInfoDto);

}

}

package com.eazybytes.loans;

…

@EnableConfigurationProperties(value = {LoansContactInfoDto.class})

…

public class LoansApplication {

public static void main(String[] args) {

SpringApplication.run(LoansApplication.class, args);

}

}

GET <http://localhost:8090/api/contact-info>

**Update Cards for Reading configuration using @ConfigurationProperties**cards\src\main\resources\application.yml

cards:

message: "Welcome to EazyBank cards related local APIs "

contactDetails:

name: "Dragos Lech - Developer"

email: "dragos@eazybank.com"

onCallSupport:

- (412) 419-3491

- (915) 382-1932

package com.eazybytes.cards.dto;

@ConfigurationProperties(prefix = "cards")

public record CardsContactInfoDto(String message, Map<String, String> contactDetails, List<String> onCallSupport) {

}

package com.eazybytes.cards.controller;

…

public class CardsController {

…

@Autowired

private CardsContactInfoDto cardsContactInfoDto;

…

@Operation(

summary = "Get Contact Info",

description = "Contact Info details that can be reached out in case of any issues"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@GetMapping("/contact-info")

public ResponseEntity<CardsContactInfoDto> getContactInfo() {

return ResponseEntity

.status(HttpStatus.OK)

.body(cardsContactInfoDto);

}

}

package com.eazybytes.cards;

…

@EnableConfigurationProperties(value = {CardsContactInfoDto.class})

…

public class CardsApplication {

public static void main(String[] args) {

SpringApplication.run(CardsApplication.class, args);

}

}

GET <http://localhost:9000/api/contact-info>

**Step -6 – Introduction to Spring Boot Profiles**

**Step -7 – Demo Spring Boot Profiles inside account microservice**

**accounts\src\main\resources\application\_qa.yml**

spring:

config:

activate:

on-profile: "qa"

build:

version: "2.0"

accounts:

message: "Welcome to EazyBank accounts related QA APIs "

contactDetails:

name: "Smitha Ray - QA Lead"

email: "smitha@eazybank.com"

onCallSupport:

- (666) 265-3765

- (666) 734-8371

**accounts\src\main\resources\application\_prod.yml**

spring:

config:

activate:

on-profile: "prod"

build:

version: "1.0"

accounts:

message: "Welcome to EazyBank accounts related prod APIs "

contactDetails:

name: "Reine Aishwarya - Product Owner"

email: "aishwarya@eazybank.com"

onCallSupport:

- (453) 392-4829

- (236) 203-0384

**accounts\src\main\resources\application.yml**

server:

port: 8080

spring:

…

config:

import:

- "application\_qa.yml"

- "application\_prod.yml"

…

GET <http://localhost:8080/api/contact-info>  
This should show you the default application.yml property values

**accounts\src\main\resources\application.yml**..

spring:

…

profiles:

active:

- "qa"

…

GET <http://localhost:8080/api/contact-info>  
This should show you the application\_qa.yml property values

**Step -8 – Externalizing configurations using command-line, JVM and environment options**

**Step -9 – Activating the profile using command-line, JVM and environment options**

**Step -10 – Add configurations inside Loans and Cards microservices  
loans\src\main\resources\application\_qa.yml**

spring:

config:

activate:

on-profile: "qa"

build:

version: "2.0"

loans:

message: "Welcome to EazyBank loans related QA APIs "

contactDetails:

name: "Smitha Ray - QA Lead"

email: "smitha@eazybank.com"

onCallSupport:

- (666) 265-3765

- (666) 734-8371

**loans\src\main\resources\application\_prod.yml**

spring:

config:

activate:

on-profile: "prod"

build:

version: "2.0"

loans:

message: "Welcome to EazyBank loans related PROD APIs "

contactDetails:

name: "Reine Aishwarya - Product Owner"

email: "aishwarya@eazybank.com"

onCallSupport:

- (453) 392-4829

- (236) 203-0384

**loans\src\main\resources\application.yml**

…

config:

import:

- "application\_qa.yml"

- "application\_prod.yml"

build:

version: "2.0"

loans:

message: "Welcome to EazyBank loans related local APIs "

contactDetails:

name: "Amaal Grega - Developer"

email: "amaal@eazybank.com"

onCallSupport:

- (452) 456-2176

- (546) 764-8934

GET http://localhost:8090/api/contact-info

This should show you the default application.yml property values

**loans\src\main\resources\application.yml**

…  
 profiles:

active:

- "qa"   
…

GET http://localhost:8090/api/contact-info

This should show you the application\_qa.yml property values

**cards\src\main\resources\application\_qa.yml**

spring:

config:

activate:

on-profile: "qa"

build:

version: "3.0"

cards:

message: "Welcome to EazyBank cards related QA APIs "

contactDetails:

name: "Cherryl Pankaj - QA Lead"

email: "cherryl@eazybank.com"

onCallSupport:

- (310) 875-4367

- (201) 236-1267

**cards\src\main\resources\application\_prod.yml**

spring:

config:

activate:

on-profile: "prod"

build:

version: "3.0"

cards:

message: "Welcome to EazyBank cards related prod APIs "

contactDetails:

name: "Sandra Harald - Product Owner"

email: "sandra@eazybank.com"

onCallSupport:

- (617) 432-2356

- (936) 564-8721

**cards\src\main\resources\application.yml**

…

config:

import:

- "application\_qa.yml"

- "application\_prod.yml"

build:

version: "3.0"

cards:

message: "Welcome to EazyBank cards related local APIs "

contactDetails:

name: "Dragos Lech - Developer"

email: "dragos@eazybank.com"

onCallSupport:

- (412) 419-3491

- (915) 382-1932

GET http://localhost:9000/api/contact-info

This should show you the default application.yml property values

**cards\src\main\resources\application.yml**

…  
 profiles:

active:

- "qa"   
…

GET http://localhost:9000/api/contact-info

This should show you the application\_qa.yml property values

**Step -11 – Drawbacks of externalized configurations using SpringBoot alone**

**Step -12 – Introduction to Spring Cloud Config**

**Step -13 – Building Config Server using Spring Cloud Config**

<properties>

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

<spring-cloud.version>2022.0.4</spring-cloud.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

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

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

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**configserver\src\main\resources\application.yml**

server:

port: 8071

package com.eazybytes.configserver;

@SpringBootApplication

@EnableConfigServer

public class ConfigserverApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigserverApplication.class, args);

}

}

**Step -14 – Reading configurations from the ClassPath location of config server**

Copy accounts\src\main\resources\application.yml to configserver\src\main\resources\config  
Copy accounts\src\main\resources\application\_qa.yml to configserver\src\main\resources\config

Copy accounts\src\main\resources\application\_prod.yml to configserver\src\main\resources\config

Rename configserver\src\main\resources\config\application.yml to accounts.yml  
Rename configserver\src\main\resources\config\application\_qa.yml to accounts-qa.yml  
Rename configserver\src\main\resources\config\application\_prod.yml to accounts-prod.yml  
  
Update accounts.yml  
…  
 config:

import:

- "accounts-qa.yml"

- "accounts-prod.yml"  
…  
  
Copy loans\src\main\resources\application.yml to configserver\src\main\resources\config  
Copy loans\src\main\resources\application\_qa.yml to configserver\src\main\resources\config

Copy loans\src\main\resources\application\_prod.yml to configserver\src\main\resources\config

Rename configserver\src\main\resources\config\application.yml to loans.yml  
Rename configserver\src\main\resources\config\application\_qa.yml to loans-qa.yml  
Rename configserver\src\main\resources\config\application\_prod.yml to loans-prod.yml  
  
Update loans.yml  
…  
 config:

import:

- "loans-qa.yml"

- "loans-prod.yml"  
…

Copy cards\src\main\resources\application.yml to configserver\src\main\resources\config  
Copy cards\src\main\resources\application\_qa.yml to configserver\src\main\resources\config

Copy ards\src\main\resources\application\_prod.yml to configserver\src\main\resources\config

Rename configserver\src\main\resources\config\application.yml to cards.yml  
Rename configserver\src\main\resources\config\application\_qa.yml to cards-qa.yml  
Rename configserver\src\main\resources\config\application\_prod.yml to cards-prod.yml  
  
Update cards.yml  
…  
 config:

import:

- "cards-qa.yml"

- "cards-prod.yml"  
…  
  
http://localhost:8071/accounts/ http://localhost:8071/accounts/qa http://localhost:8071/accounts/prod

http://localhost:8071/loans/ http://localhost:8071/loans/qa http://localhost:8071/loans/prod

http://localhost:8071/cards/ http://localhost:8071/cards/qa http://localhost:8071/cards/prod

**Step -15 – Updating Accounts Microservice to read properties from Config Server**

Delete all profile related files from accounts microservice

accounts\src\main\resources\application\_qa.yml  
accounts\src\main\resources\application\_prod.yml

Delete the config related configuration from application.yml about these config properties  
 ~~config:~~

~~import:~~

~~- "application\_qa.yml"~~

~~- "application\_prod.yml"~~

~~profiles:~~

~~active:~~

~~- "qa"~~

~~build:~~

~~version: "1.0"~~

~~accounts:~~

~~message: "Welcome to EazyBank accounts related local APIs "~~

~~contactDetails:~~

~~name: "John Doe - Developer"~~

~~email: "john@eazybank.com"~~

~~onCallSupport:~~

~~- (555) 555-1234~~

~~- (555) 523-1345~~

Give the name to the application in application.yml  
accounts\src\main\resources\application.yml  
…

spring:

application:

name: "accounts"  
 profiles:

active: "prod"

…  
This name “accounts” should match the config file name in config microservice  
configserver\src\main\resources\config\accounts.yml  
configserver\src\main\resources\config\accounts-prod.yml  
configserver\src\main\resources\config\accounts-qa.yml  
  
Add dependency of the spring config client in account microservice  
<properties>

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

<spring-cloud.version>2022.0.4</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>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

Update application.yml of account microservice  
accounts\src\main\resources\application.yml  
..

spring:

..

config:

import: "configserver:http://localhost:8071/"

This is an indication to the accounts application that it needs to connect to the config server.

import:"**configserver**:http://localhost:8071/"  
This value **configserver** should match   
spring:

application:

name: "**configserver**"  
  
We should also add **optional**  
…

config:

import: "**optional**:configserver:http://localhost:8071/"  
This means if accounts application fails to connect to the config server it should still start the accounts application and not fail.  
  
You should add optional if your accounts microservice can perform without config server  
You should NOT add optional if your accounts microservice cannot perform without config server

Start config server 🡺 account service  
  
GET <http://localhost:8080/api/contact-info>  
This should get the prod configuration from configserver\src\main\resources\config\accounts-prod.yml because we have configure **prod** in  
accounts\src\main\resources\application.ymlspring:

application:

name: "accounts"

profiles:

active: "prod"

**Step -16 – Updating Loans and Cards Microservice to read properties from Config Server**

**Update Loans Microservice**Delete all profile related files from accounts microservice

loans\src\main\resources\application\_qa.yml  
loans\src\main\resources\application\_prod.yml

Delete the config related configuration from application.yml about these config properties  
 ~~config:~~

~~import:~~

~~- "application\_qa.yml"~~

~~- "application\_prod.yml"~~

~~profiles:~~

~~active:~~

~~- "qa"~~

~~build:~~

~~version: "2.0"~~

~~loans:~~

~~message: "Welcome to EazyBank loans related local APIs "~~

~~contactDetails:~~

~~name: "Amaal Grega - Developer"~~

~~email: "amaal@eazybank.com"~~

~~onCallSupport:~~

~~- (452) 456-2176~~

~~- (546) 764-8934~~

Give the name to the application in application.yml  
loans src\main\resources\application.yml  
…

spring:

application:

name: " loans"  
 profiles:

active: "prod"

…  
This name “loans” should match the config file name in config microservice  
configserver\src\main\resources\config\loans.yml  
configserver\src\main\resources\config\loans-prod.yml  
configserver\src\main\resources\config\loans-qa.yml  
  
Add dependency of the spring config client in loans microservice  
<properties>

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

<spring-cloud.version>2022.0.4</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>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

Update application.yml of loans microservice  
loans\src\main\resources\application.yml  
..

spring:

..

config:

import: "configserver:http://localhost:8071/"

This is an indication to the loans application that it needs to connect to the config server.

import:"**configserver**:http://localhost:8071/"  
This value **configserver** should match   
spring:

application:

name: "**configserver**"  
  
We should also add **optional**  
…

config:

import: "**optional**:configserver:http://localhost:8071/"  
This means if loans application fails to connect to the config server it should still start the loans application and not fail.  
  
You should add optional if your loans microservice can perform without config server  
You should NOT add optional if your loans microservice cannot perform without config server

Start config server 🡺 loans service  
  
GET <http://localhost:8090/api/contact-info>  
This should get the prod configuration from configserver\src\main\resources\config\loans-prod.yml because we have configure **prod** in  
loans\src\main\resources\application.ymlspring:

application:

name: "loans"

profiles:

active: "prod"

**Update Cards Microservice**Delete all profile related files from accounts microservice

cards\src\main\resources\application\_qa.yml  
cards\src\main\resources\application\_prod.yml

Delete the config related configuration from application.yml about these config properties  
 ~~config:~~

~~import:~~

~~- "application\_qa.yml"~~

~~- "application\_prod.yml"~~

~~profiles:~~

~~active:~~

~~- "qa"~~

~~build:~~

~~version: "3.0"~~

~~cards:~~

~~message: "Welcome to EazyBank cards related local APIs "~~

~~contactDetails:~~

~~name: "Dragos Lech - Developer"~~

~~email: "dragos@eazybank.com"~~

~~onCallSupport:~~

~~- (412) 419-3491~~

~~- (915) 382-1932~~

Give the name to the application in application.yml  
cards src\main\resources\application.yml  
…

spring:

application:

name: " cards"  
 profiles:

active: "prod"

…  
This name “cards” should match the config file name in config microservice  
configserver\src\main\resources\config\cards.yml  
configserver\src\main\resources\config\cards-prod.yml  
configserver\src\main\resources\config\cards-qa.yml  
  
Add dependency of the spring config client in cards microservice  
<properties>

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

<spring-cloud.version>2022.0.4</spring-cloud.version>

</properties>

<dependency>

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

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

</dependency>

<dependencyManagement>

<dependencies>

<dependency>

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

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

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

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

Update application.yml of cards microservice  
cards\src\main\resources\application.yml  
..

spring:

..

config:

import: "configserver:http://localhost:8071/"

This is an indication to the cards application that it needs to connect to the config server.

import:"**configserver**:http://localhost:8071/"  
This value **configserver** should match   
spring:

application:

name: "**configserver**"  
  
We should also add **optional**  
…

config:

import: "**optional**:configserver:http://localhost:8071/"  
This means if cards application fails to connect to the config server it should still start the cards application and not fail.  
  
You should add optional if your cards microservice can perform without config server  
You should NOT add optional if your cards microservice cannot perform without config server

Start config server 🡺 cards service  
  
GET <http://localhost:9000/api/contact-info>  
This should get the prod configuration from configserver\src\main\resources\config\cards-prod.yml because we have configure **prod** in  
cards\src\main\resources\application.ymlspring:

application:

name: "cards"

profiles:

active: "prod"

**Step -17 – Reading configurations from a file system location**

Copy all files inside configserver\src\main\resources\config\  
to folder  
D:\Nilesh\github\eazybytes\_ms\_k8s\config

Update configserver\src\main\resources\application.yml  
spring:

application:

name: "configserver"

profiles:

active: native

cloud:

config:

server:

native:

# search-locations: "classpath:/config"

search-locations: D:\Nilesh\github\eazybytes\_ms\_k8s\config

…

Start config server 🡪 accounts 🡪 loans 🡪 cards  
  
GET <http://localhost:8080/api/contact-info>  
GET <http://localhost:8090/api/contact-info>

GET <http://localhost:9000/api/contact-info>

The endpoints should should show the properties values from files located at  
D:\Nilesh\github\eazybytes\_ms\_k8s\config  
  
**Step -18 – Reading configurations from GitHub repository**

Create repository on github  
<https://github.com/nileshzarkar/config>  
  
Upload all files in D:\Nilesh\github\eazybytes\_ms\_k8s\config folder to this repository  
Note: Update the value of some properties to identify the properties are coming from github  
  
Update **configserver\src\main\resources\application.yml**spring:

application:

name: "configserver"

profiles:

# active: native

active: git

cloud:

config:

server:

# native:

# search-locations: "classpath:/config"

# search-locations: D:\Nilesh\github\eazybytes\_ms\_k8s\config

git:

uri: https://github.com/nileshzarkar/config.git

default-label: main

timeout: 5

clone-on-start: true

force-pull: true

server:

port: 8071

Start config server 🡪 accounts 🡪 loans 🡪 cards  
  
GET <http://localhost:8080/api/contact-info>  
GET <http://localhost:8090/api/contact-info>

GET <http://localhost:9000/api/contact-info>

**Step -19 – Encryption and Decryption of properties inside Config Server**

configserver\src\main\resources\application.yml

The endpoints should should show the properties values from files located at  
<https://github.com/nileshzarkar/config.git>

spring:

…

encrypt:

key: "123456789"

…

Star the config server  
POST <http://localhost:8071/encrypt>

RAW TEXT   
aishwarya@eazybank.com  
Response  
bf538d958baca10a5e973e8098b4d2e80c1b7a21d3fbb987b58bae2a57ac99f195111085453a38d6bf3382d15dc3f0da  
  
<https://github.com/nileshzarkar/config/blob/main/accounts-prod.yml>  
Update the email value from plain text to encrypted value  
email: [aishwarya@eazybank.com](mailto:aishwarya@eazybank.com)

to   
email: "{cipher}bf538d958baca10a5e973e8098b4d2e80c1b7a21d3fbb987b58bae2a57ac99f195111085453a38d6bf3382d15dc3f0da"

If the config server able to decrypt the encrypted value  
POST <http://localhost:8071/decrypt>  
RAW TEXT   
bf538d958baca10a5e973e8098b4d2e80c1b7a21d3fbb987b58bae2a57ac99f195111085453a38d6bf3382d15dc3f0da

Response  
aishwarya@eazybank.com

Check whether accounts is getting the decrypted value  
GET <http://localhost:8080/api/contact-info>

**Update the Loans Microservice for encrypt and decrypt**

configserver\src\main\resources\application.yml

The endpoints should should show the properties values from files located at  
<https://github.com/nileshzarkar/config.git>

spring:

…

encrypt:

key: "123456789"

…

Star the config server  
POST <http://localhost:8071/encrypt>

RAW TEXT   
aishwarya@eazybank.com  
Response  
bf538d958baca10a5e973e8098b4d2e80c1b7a21d3fbb987b58bae2a57ac99f195111085453a38d6bf3382d15dc3f0da  
  
<https://github.com/nileshzarkar/config/blob/main/loans-prod.yml>  
Update the email value from plain text to encrypted value  
email: [aishwarya@eazybank.com](mailto:aishwarya@eazybank.com)

to   
email: "{cipher}bf538d958baca10a5e973e8098b4d2e80c1b7a21d3fbb987b58bae2a57ac99f195111085453a38d6bf3382d15dc3f0da"

If the config server able to decrypt the encrypted value  
POST <http://localhost:8071/decrypt>  
RAW TEXT   
bf538d958baca10a5e973e8098b4d2e80c1b7a21d3fbb987b58bae2a57ac99f195111085453a38d6bf3382d15dc3f0da

Response  
aishwarya@eazybank.com

Check whether loans is getting the decrypted value  
GET <http://localhost:8090/api/contact-info>

**Update the Cards Microservice for encrypt and decrypt**

configserver\src\main\resources\application.yml

The endpoints should should show the properties values from files located at  
<https://github.com/nileshzarkar/config.git>

spring:

…

encrypt:

key: "123456789"

…

Star the config server  
POST <http://localhost:8071/encrypt>

RAW TEXT   
sandra@eazybank.com  
Response  
61de35bd288fc14b880ce5afca0fcd00357f375035721fdca9ee29c0c11821c5ec215fe9a9315743522b55106dacb1f9  
  
<https://github.com/nileshzarkar/config/blob/main/cards-prod.yml>  
Update the email value from plain text to encrypted value  
email: sandra@eazybank.com

to   
email: " {cipher} 61de35bd288fc14b880ce5afca0fcd00357f375035721fdca9ee29c0c11821c5ec215fe9a9315743522b55106dacb1f9"

If the config server able to decrypt the encrypted value  
POST <http://localhost:8071/decrypt>  
RAW TEXT   
61de35bd288fc14b880ce5afca0fcd00357f375035721fdca9ee29c0c11821c5ec215fe9a9315743522b55106dacb1f9

Response  
sandra@eazybank.com

Check whether cards is getting the decrypted value  
GET <http://localhost:9000/api/contact-info>

**Step -20 – Refresh configuration at runtime using refresh actuator path**

Make sure you have actuator dependency in all accounts, loans, cards and config-server

<dependency>

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

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

</dependency>

~~public record AccountsContactInfoDto(String message, Map<String, String> contactDetails, List<String> onCallSupport) { }~~

package com.eazybytes.accounts.dto;

@ConfigurationProperties(prefix = "accounts")

@Getter @Setter

public class AccountsContactInfoDto {

private String message;

private Map<String, String> contactDetails;

private List<String> onCallSupport;

}

Update accounts\src\main\resources\application.yml

…  
management:

endpoints:

web:

exposure:

include: "\*"

~~public record LoansContactInfoDto(String message, Map<String, String> contactDetails, List<String> onCallSupport) {~~

~~}~~

package com.eazybytes.loans.dto;

@ConfigurationProperties(prefix = "loans")

@Getter @Setter

public class LoansContactInfoDto {

private String message;

private Map<String, String> contactDetails;

private List<String> onCallSupport;

}

Update loans\src\main\resources\application.yml

…  
management:

endpoints:

web:

exposure:

include: "\*"

~~public record CardsContactInfoDto(String message, Map<String, String> contactDetails, List<String> onCallSupport) {~~

~~}~~

package com.eazybytes.cards.dto;

@ConfigurationProperties(prefix = "cards")

@Getter @Setter

public class CardsContactInfoDto {

private String message;

private Map<String, String> contactDetails;

private List<String> onCallSupport;

}

Update cards\src\main\resources\application.yml

…  
management:

endpoints:

web:

exposure:

include: "\*"

Start config-server (wait until it starts successfully) 🡪 accounts 🡪 loans 🡪 cards

Check what configuration value each microservice is getting from config server  
GET <http://localhost:8080/api/contact-info>  
GET <http://localhost:8090/api/contact-info>  
GET <http://localhost:9000/api/contact-info>  
  
Update the configuration files in git-hub  
<https://github.com/nileshzarkar/config/blob/main/accounts-prod.yml>  
…  
message: "Welcome to EazyBank accounts related prod APIs GIT-HUB REFRESH-TEST"

…  
<https://github.com/nileshzarkar/config/blob/main/loans-prod.yml>

…  
message: "Welcome to EazyBank loans related PROD APIs GIT-HUB REFRESH-TEST"

…  
<https://github.com/nileshzarkar/config/blob/main/cards-prod.yml>

…  
message: "Welcome to EazyBank cards related prod APIs GIT-HUB REFRESH-TEST"  
…

Config server can get the updated values  
<http://localhost:8071/accounts/prod>  
<http://localhost:8071/loans/prod>  
http://localhost:8071/cards/prod

Check what configuration value each microservice is getting from config server  
GET <http://localhost:8080/api/contact-info>  
GET <http://localhost:8090/api/contact-info>  
GET <http://localhost:9000/api/contact-info>

You will NOT GET updated values in the service endpoints

**Refresh accounts config**Go the actuator endpoint of the accounts microservice  
<http://localhost:8080/actuator>  
Look for refresh actuator endpoint

<http://localhost:8080/actuator/refresh> This is a POST method  
  
POST <http://localhost:8080/actuator/refresh>  
NO BODY  
  
Check what configuration value accounts microservice is getting from config server  
GET <http://localhost:8080/api/contact-info>  
You will GET updated values in the service endpoints

**Refresh loans config**Go the actuator endpoint of the loans microservice  
<http://localhost:8090/actuator>  
Look for refresh actuator endpoint

<http://localhost:8090/actuator/refresh> This is a POST method  
  
POST <http://localhost:8090/actuator/refresh>  
NO BODY  
  
Check what configuration value loans microservice is getting from config server  
GET <http://localhost:8090/api/contact-info>  
You will GET updated values in the service endpoints

**Refresh cards config**Go the actuator endpoint of the cards microservice  
<http://localhost:9000/actuator>  
Look for refresh actuator endpoint

<http://localhost:9000/actuator/refresh> This is a POST method  
  
POST <http://localhost:9000/actuator/refresh>  
NO BODY  
  
Check what configuration value cards microservice is getting from config server  
GET <http://localhost:9000/api/contact-info>  
You will GET updated values in the service endpoints

**Step -21 – Refresh configurations at runtime using Spring Cloud Bus**

Install Rabbit MQ on your local machine

<https://www.rabbitmq.com/download.html>  
  
docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.12-management

Install Rabbit MQ on your local machine

Update accounts pom.xml

<dependency>

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

<artifactId>spring-cloud-starter-bus-amqp</artifactId>

</dependency>

Update accounts application.yml  
…  
 rabbitmq:

host: "localhost"

port: 5672

username: "guest"

password: "guest"  
…

Update loans pom.xml

<dependency>

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

<artifactId>spring-cloud-starter-bus-amqp</artifactId>

</dependency>

Update loans application.yml  
…  
 rabbitmq:

host: "localhost"

port: 5672

username: "guest"

password: "guest"  
…

Update cards pom.xml

<dependency>

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

<artifactId>spring-cloud-starter-bus-amqp</artifactId>

</dependency>

Update cards application.yml  
…  
 rabbitmq:

host: "localhost"

port: 5672

username: "guest"

password: "guest"  
…

Start config-server (wait until it starts successfully) 🡪 accounts 🡪 loans 🡪 cards

Check what configuration value each microservice is getting from config server  
GET <http://localhost:8080/api/contact-info>  
GET <http://localhost:8090/api/contact-info>  
GET <http://localhost:9000/api/contact-info>  
  
Update the configuration files in git-hub  
<https://github.com/nileshzarkar/config/blob/main/accounts-prod.yml>  
…  
message: "Welcome to EazyBank accounts related prod APIs GIT-HUB "

…  
<https://github.com/nileshzarkar/config/blob/main/loans-prod.yml>

…  
message: "Welcome to EazyBank loans related PROD APIs GIT-HUB"

…  
<https://github.com/nileshzarkar/config/blob/main/cards-prod.yml>

…  
message: "Welcome to EazyBank cards related prod APIs GIT-HUB"  
…

Config server can get the updated values  
<http://localhost:8071/accounts/prod>  
<http://localhost:8071/loans/prod>  
http://localhost:8071/cards/prod

Check what configuration value each microservice is getting from config server  
GET <http://localhost:8080/api/contact-info>  
GET <http://localhost:8090/api/contact-info>  
GET <http://localhost:9000/api/contact-info>

You will NOT GET updated values in the service endpoints

**Refresh accounts config**Go the actuator endpoint of the accounts microservice  
<http://localhost:8080/actuator>  
Look for refresh actuator endpoint

http://localhost:8080/actuator/busrefresh This is a POST method  
  
POST http://localhost:8080/actuator/busrefresh  
NO BODY  
  
Check what configuration value accounts microservice is getting from config server  
GET <http://localhost:8080/api/contact-info>  
GET <http://localhost:8090/api/contact-info>  
GET <http://localhost:9000/api/contact-info>  
You will GET updated values in the service endpoints

**Step -22 – Refresh config at runtime using Spring Cloud Bus & Spring Cloud monitor**

**Step -23 – Updating docker compose file to adapt Config Server changes – Part 1**

**Step -24 – Introduction to Liveness and Readiness**

**Step -25 – Updating docker compose file to adapt Config Server changes – Part 2**

**Step -26 – Optimizing Docker compose file**

**Step -27 – Generating Docker images and pushing them into Docker hub**

**Step -28 – Testing Config Server changes end to end using Docker Compose and default profile**

**Step -29 – Preparing Docker Compose files for QA and prod profiles**