**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

**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>  
  
**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>

**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"

profiles:

active:

- "qa"

…

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  
  
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**

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

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

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

**Step -18 – Reading configurations from GitHub repository**

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

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

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

**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**