개발가이드

# 사전준비

## 1. Knowledge에 참고자료 추가

### 1) 개발주석표준

**제목: 개발주석표준**

| **주석 표준을 준수하면 코드의 유지보수성과 가독성이 향상되어 팀원 간의 협업이 원활해집니다.**  **코드 작성 시 아래 주석 표준을 준수해 주세요.**  **---**  \*\*개발 주석 표준\*\*  1. \*\*주석의 목적 명확화\*\*  - \*\*왜\*\* 이 코드를 작성했는지, \*\*어떻게\*\* 동작하는지 설명합니다.  - 복잡한 로직이나 특별한 의도가 있는 부분에 주석을 작성합니다.  2. \*\*일관된 스타일 사용\*\*  - 팀 내에서 합의된 주석 스타일 가이드라인을 따릅니다.  - 프로그래밍 언어별 표준 주석 문법을 사용합니다.  3. \*\*명확하고 간결한 표현\*\*  - 간결한 문장으로 핵심을 전달합니다.  - 이해하기 쉬운 언어와 용어를 사용합니다.  4. \*\*코드와 주석의 동기화\*\*  - 코드 변경 시 관련 주석도 함께 업데이트합니다.  - 오래되거나 불필요한 주석은 제거합니다.  5. \*\*주석의 위치와 종류\*\*  - \*\*파일 헤더 주석\*\*: 파일의 목적, 작성자, 버전 정보를 포함합니다.  - \*\*클래스/모듈 주석\*\*: 클래스나 모듈의 역할과 사용법을 설명합니다.  - \*\*메서드/함수 주석\*\*: 기능, 입력 파라미터, 반환 값, 예외 사항을 기술합니다.  - \*\*인라인 주석\*\*: 코드 내 특정 라인이나 블록에 대한 추가 설명이 필요할 때 사용합니다.  6. \*\*표준화된 주석 태그 사용\*\*  - `@param`, `@return`, `@throws` 등의 태그를 활용하여 자동 문서화에 도움을 줍니다.  - `TODO`, `FIXME` 등을 사용하여 향후 작업이나 수정 사항을 표시합니다.  7. \*\*적절한 언어 사용\*\*  - 팀의 공용어를 사용하여 모든 팀원이 이해할 수 있게 합니다.  - 전문적이고 예의 바른 표현을 사용하며, 비속어나 개인적인 의견은 피합니다.  8. \*\*예제\*\*  ```java  /\*\*  \* 사용자 정보를 관리하는 클래스입니다.  \*  \* @author  \* @version 1.0  \*/  public class UserManager {  /\*\*  \* 사용자 ID로 사용자 정보를 조회합니다.  \*  \* @param userId 조회할 사용자 ID  \* @return 사용자 객체를 반환합니다. 해당 ID의 사용자가 없으면 null을 반환합니다.  \* @throws IllegalArgumentException userId가 null이거나 빈 문자열인 경우  \*/  public User getUserById(String userId) throws IllegalArgumentException {  // 입력된 사용자 ID의 유효성 검사  if (userId == null || userId.isEmpty()) {  throw new IllegalArgumentException("사용자 ID는 null이거나 빈 문자열일 수 없습니다.");  }  // TODO: 데이터베이스에서 사용자 정보를 조회하는 로직 구현 필요  return null;  }  }  ```  9. \*\*자동 생성 주석의 활용\*\*  - IDE나 도구에서 자동으로 생성되는 주석은 필요에 따라 수정하거나 보완합니다.  - 불필요한 자동 생성 주석은 제거하여 코드의 가독성을 높입니다.  10. \*\*주석 작성 시 주의 사항\*\*  - 민감한 정보나 보안과 관련된 내용을 주석에 포함하지 않습니다.  - 주석은 코드의 동작을 대체하지 않으며, 오히려 보완해야 합니다. |
| --- |

### 2) Sample 소스

**제목: sample\_project.java**

| // settings.gradle  rootProject.name = 'cms'  include 'common'  include 'authentication'  include 'content-management'  include 'token-management'  // build.gradle  plugins {  id 'org.springframework.boot' version '3.4.0' apply false  id 'io.spring.dependency-management' version '1.1.4' apply false  id 'java'  }  subprojects {  apply plugin: 'java'  apply plugin: 'org.springframework.boot'  apply plugin: 'io.spring.dependency-management'    group = 'com.telecom.cms'  version = '0.0.1-SNAPSHOT'  sourceCompatibility = '21'    repositories {  mavenCentral()  }    dependencies {  // Spring Boot Starters  implementation 'org.springframework.boot:spring-boot-starter'  implementation 'org.springframework.boot:spring-boot-starter-web'  implementation 'org.springframework.boot:spring-boot-starter-validation'  implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  implementation 'org.springframework.boot:spring-boot-starter-security'  implementation 'org.springframework.boot:spring-boot-starter-actuator'    // Database  runtimeOnly 'org.postgresql:postgresql'    // Swagger  implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.4.0'    // JWT  implementation 'io.jsonwebtoken:jjwt-api:0.11.5'  runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.5'  runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.5'    // Lombok  compileOnly 'org.projectlombok:lombok'  annotationProcessor 'org.projectlombok:lombok'    // Test  testImplementation 'org.springframework.boot:spring-boot-starter-test'  testImplementation 'org.springframework.security:spring-security-test'  }    test {  useJUnitPlatform()  }  }  // token-management/build.gradle  dependencies {  implementation project(':common')    // Azure Blob Storage & SAS Token 관련  implementation 'com.azure:azure-storage-blob:12.22.1'  implementation 'com.azure:azure-storage-common:12.21.1'  }  bootJar {  archiveFileName = "token-management.jar"  }  // token-management/src/main/resources/application.yml  server:  port: ${SERVER\_PORT:8082}  servlet:  context-path: /  allowed-origins: ${ALLOWED\_ORIGINS:http://localhost:3000}  spring:  application:  name: token-management-service  datasource:  url: jdbc:postgresql://${POSTGRES\_HOST:localhost}:${POSTGRES\_PORT:5432}/${POSTGRES\_DB:tokendb}  username: ${POSTGRES\_USER:postgres}  password: ${POSTGRES\_PASSWORD:postgres}  jpa:  hibernate:  ddl-auto: ${JPA\_DDL\_AUTO:update}  show-sql: ${JPA\_SHOW\_SQL:true}  properties:  hibernate:  dialect: org.hibernate.dialect.PostgreSQLDialect  format\_sql: true  azure:  storage:  connection-string: ${AZURE\_STORAGE\_CONNECTION\_STRING}  container:  marketing-materials: ${AZURE\_CONTAINER\_MARKETING:marketing-materials}  user-guides: ${AZURE\_CONTAINER\_USER\_GUIDES:user-guides}  token:  default-validity-minutes: ${TOKEN\_DEFAULT\_VALIDITY:30}  cleanup-cron: ${TOKEN\_CLEANUP\_CRON:0 0 \* \* \* \*}  springdoc:  swagger-ui:  path: /swagger-ui.html  operations-sorter: method  api-docs:  path: /api-docs  logging:  level:  com.telecom.cms.token: ${LOG\_LEVEL:DEBUG}  // token-management/src/main/java/com/telecom/cms/token/config/SecurityConfig  package com.telecom.cms.token.config;  import lombok.RequiredArgsConstructor;  import org.springframework.beans.factory.annotation.Value;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.security.config.annotation.web.builders.HttpSecurity;  import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;  import org.springframework.security.config.http.SessionCreationPolicy;  import org.springframework.security.web.SecurityFilterChain;  import org.springframework.web.client.RestTemplate;  import org.springframework.web.cors.CorsConfiguration;  import org.springframework.web.cors.CorsConfigurationSource;  import org.springframework.web.cors.UrlBasedCorsConfigurationSource;  import java.util.Arrays;  import java.util.List;  import java.util.stream.Collectors;  @Configuration  @EnableWebSecurity  @RequiredArgsConstructor  public class SecurityConfig {  @Value("${server.allowed-origins}")  private String allowedOrigins;  /\*\*  \* RestTemplate Bean을 생성합니다.  \*  \* @return RestTemplate 인스턴스  \*/  @Bean  public RestTemplate restTemplate() {  return new RestTemplate();  }  @Bean  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  http  .csrf(AbstractHttpConfigurer::disable)  .cors(cors -> cors.configurationSource(corsConfigurationSource()))  .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))  .authorizeHttpRequests(auth -> auth  .requestMatchers("/api/tokens/\*\*", "/swagger-ui/\*\*", "/swagger-ui.html", "/api-docs/\*\*", "/actuator/\*\*").permitAll()  .anyRequest().authenticated()  );  return http.build();  }  @Bean  public CorsConfigurationSource corsConfigurationSource() {  CorsConfiguration configuration = new CorsConfiguration();  List<String> origins = Arrays.stream(allowedOrigins.split(","))  .map(String::trim)  .filter(origin -> !origin.isEmpty())  .collect(Collectors.toList());  configuration.setAllowedOriginPatterns(origins);  configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"));  configuration.setAllowedHeaders(Arrays.asList("authorization", "content-type", "x-auth-token"));  configuration.setExposedHeaders(Arrays.asList("x-auth-token"));  configuration.setAllowCredentials(true);  configuration.setMaxAge(3600L);  UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  source.registerCorsConfiguration("/\*\*", configuration);  return source;  }  }  // token-management/src/main/java/com/telecom/cms/token/config/SwaggerConfig  package com.telecom.cms.token.config;  import io.swagger.v3.oas.models.Components;  import io.swagger.v3.oas.models.OpenAPI;  import io.swagger.v3.oas.models.info.Info;  import io.swagger.v3.oas.models.security.SecurityRequirement;  import io.swagger.v3.oas.models.security.SecurityScheme;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  /\*\*  \* Swagger 설정 클래스입니다.  \*/  @Configuration  public class SwaggerConfig {  /\*\*  \* Swagger OpenAPI 구성을 제공합니다.  \*  \* @return OpenAPI 인스턴스  \*/  @Bean  public OpenAPI openAPI() {  String securitySchemeName = "bearerAuth";  return new OpenAPI()  .info(new Info()  .title("토큰 관리 서비스 API")  .description("통신사 구독 서비스의 토큰 관리 API 문서")  .version("v1.0.0"))  .addSecurityItem(new SecurityRequirement().addList(securitySchemeName))  .components(new Components()  .addSecuritySchemes(securitySchemeName,  new SecurityScheme()  .name(securitySchemeName)  .type(SecurityScheme.Type.HTTP)  .scheme("bearer")  .bearerFormat("JWT")));  }  } |
| --- |

**제목: authentication.java**

| **JWT기반 인증 서비스 샘플입니다.**  **아래 파일의 내용을 올립니다.**  [**https://drive.google.com/file/d/1ieP6oqlEfG7CM-hk21bUkOPzi7wDCWU3/view?usp=drive\_link**](https://drive.google.com/file/d/1ieP6oqlEfG7CM-hk21bUkOPzi7wDCWU3/view?usp=drive_link) |
| --- |

## 2. Instruction에 개발 지시 사항 추가

| [개발]  'c:'로 시작하면 개발 요청임.  {패키지구조 설계 요구사항}  <요청사항>  - Knowledge의 '패키지 구조도'기반으로 설계  - 필요 시 클래스를 추가하고 그 이유를 제시  - Clean아키텍처 적용 시 Port/Adapter라는 용어 대신 Clean 아키텍처에 맞는 용어 사용  - 클래스를 포함한 전체 패키지 구조도를 한꺼번에 제공  <참고자료>  - 클래스 설계서  - 패키지 구조도  <응답형식>  코드블록  {백엔드 개발 요구사항}  <요청사항>  - Spring Boot 3.4.0으로 개발  - Java 21과 Gradle로 개발  - 멀티 프로젝트로 개발  - 설계된 패키지 구조도와 동일하게 개발  - 'Class 설계서'의 설계와 일치하도록 개발  - 지정한 {개발방식}으로 개발  - application.yml의 모든 값은 환경변수를 이용  - 모든 서비스에서 사용하는 공통 라이브러리는 Root project의 build.gradle에 지정  - Swagger Page에 한글로 API 사용법 제공  - Swagger library는 'org.springdoc' 사용  - Knowledge의 '개발주석표준'을 준수하여 주석을 작성  - 각 소스 상단에 전체 경로를 '// '뒤에 표시  예) // build.gradle  예) // query/build.gradle  예) // query/src/main/resources/application.yml  예) // query/src/main/java/com/kt/usage/query/controller/UsageQueryController.java  - 인증이 필요한 경우 'authentication.java'를 참조하여 인증 관련 설정과 클래스 개발  - Lombok을 사용하므로 필요 없는 생성자, Getter, Setter는 생략  <참고자료>  - Userstory.pdf  - API 설계서  - 시퀀스 설계서  - 클래스 설계서  - 패키지 구조도  - 데이터 설계서  - 개발주석표준  <예시>  - sample\_project  <응답순서>  아래 순서로 하나의 코드블록에 한꺼번에 결과를 제공  1.아래는 단일 코드블록으로 제공  - Root 프로젝트의 settings.gradle  - Root 프로젝트의 build.gradle  - Root 프로젝트의 .gitignore  - 각 서비스의 build.gradle  - 각 서비스의 application.yml  2.아래는 각 서비스마다 단일 코드블록으로 제공  - 각 서비스의 클래스들  3.common 서비스를 단일 코드블록으로 제공  {프론트엔드 개발 요구사항}  <요청사항>  - 'Userstory'를 꼼꼼히 파악하여 개발  - 'MergedSource'에 있는 각 서비스의 Controller 클래스에 있는 API 연동  - \*모바일 화면 크기에 맞게 "넓이"와 "높이"를 최적화\*  - \*이미지 크기는 화면에 맞게 적절하게 조정\*  - \*React 18이상 버전\*으로 개발해야 함  - React 버전에 맞는 Matrial UI 버전을 사용하여 개발함  - 이미지 파일은 public/images폴더 하위에 위치하게 하고, 애플리케이션에서 이 이미지를 참조하도록 함  - 금액은 숫자로만 표시하고 3자리마다 콤마를 표시함  - 각 화면 상단 좌측에 이전화면으로 돌아가는 Back 아이콘 버튼과 화면 타이틀 표시  - API 호출 시 host는 물리아키텍처를 참조(설계에 따라 Gateway Host 한개이거나, 각 서비스별로 다를 수 있음)  - login 외 다른 api는 login API에서 받은 JWT토큰을 'Authorization' 헤더에 추가해야 함  - {ingress host} 값을 이용하여 아래 예와 같이 필요한 API 주소를 public폴더 하위에 runtime-env.js파일로 생성함(ingress host값이 없으면 20.1.2.3을 사용)  ```  window.\_\_runtime\_config\_\_ = {  AUTH\_URL: 'http://{ingress host}/auth',  CONTENT\_URL: 'http://{ingress host}/content',  TOKEN\_URL: 'http://{ingress host}/token'  }  ```  - 각 소스 상단에 전체 경로를 '//\* '뒤에 표시  예) //\* package.json  예) //\* public/index.html  예) //\* src/index.js  예) //\* src/pages/Login.js  <참고자료>  - Userstory.pdf  - API 설계서  - MergedSource  <응답순서>  - 프로젝트 구조, 프로젝트 실행 방법을 먼저 제시  - 프로젝트 구조에 대한 확인을 받고 각 파일 소스 개발 시작  - 각 파일 소스는 분리하지 말고 \*\*단일 코드블록\*\*으로 제공  - App.js, App.css, index.js, index.css, index.html, package.json, .gitignore, manifest.json도 누락하지 말고 보여줄 것  <응답형식>  코드블록 |
| --- |

# 개발

## 1. 코딩

| c: Knowledge의 '패키지 구조도'를 기반으로 개발에 필요한 클래스를 포함한 패키지 구조도를 보여주세요. 필요 시 클래스를 추가해 주세요. |
| --- |

패키지 구조 검토 완료 후 개발 시작

| c: 백엔드 개발을 시작해 주세요. 개발방식은 {WebMVC 또는 WebFlux}를 사용하세요.  예)  c: 백엔드 개발을 시작해 주세요. 개발 방식은 WebMVC를 사용하세요. |
| --- |

프론트엔드 프롬프트는 아래와 같이 합니다.

| c: 프론트엔드 개발을 시작해 주세요. |
| --- |

만약, 설계된 화면이 있다면 Knowledge에 이미지를 올리고 아래와 같이 하세요.

| c: 프론트엔드 개발을 시작해 주세요.  첨부한 이미지를 참고하여 개발해 주세요. |
| --- |

기존에 참조할 프로젝트 구조가 있다면 예시로 제시하세요.

| c: 프론트엔드 개발을 시작해 주세요.  - 첨부한 이미지를 참고하여 개발  - 프로젝트 구조는 예시 참조  [프로젝트 구조 예시]  lifesub-web/  ├── public/  │ ├── images/  │ │ ├── logo192.png  │ │ ├── logo512.png  │ │ ├── liker.png  │ │ ├── collector.png  │ │ └── addict.png  │ ├── index.html  │ ├── manifest.json  │ └── runtime-env.js  ├── src/  │ ├── components/  │ │ ├── auth/  │ │ │ └── LoginForm.js  │ │ ├── common/  │ │ │ ├── ErrorMessage.js  │ │ │ ├── Header.js  │ │ │ ├── Layout.js  │ │ │ ├── LoadingSpinner.js  │ │ │ └── SubscriptionCard.js  │ │ ├── main/  │ │ │ ├── MySubscriptions.js  │ │ │ ├── RecommendCategory.js  │ │ │ └── TotalFee.js  │ │ └── subscriptions/  │ │ ├── CategoryList.js  │ │ ├── SubscriptionDetail.js  │ │ └── SubscriptionList.js  │ ├── contexts/  │ │ └── AuthContext.js  │ ├── pages/  │ │ ├── LoginPage.js  │ │ ├── MainPage.js  │ │ ├── SubscriptionDetailPage.js  │ │ └── SubscriptionListPage.js  │ ├── services/  │ │ └── api.js  │ ├── utils/  │ │ └── formatters.js  │ ├── App.css  │ ├── App.js  │ ├── index.css  │ └── index.js  └── package.json |
| --- |

## 2. 백엔드 프로젝트 생성

### 1) 프로젝트 생성

클로드가 생성한 소스를 프로젝트 구조로 변환합니다.

* 아래 링크에서 genprj.py 파일을 c:\home\workspace디렉토리에 다운로드   
  <https://drive.google.com/file/d/1jorBH7r1BAlDsW89b_33ThRDVF5CL54e/view?usp=drive_link>
* MobaXTerm에서 Local Ubuntu를 열고 홈으로 이동: cd 라고 치고 엔터 치면 됨
* 클로드가 제공한 소스를 한 파일로 합침: 예) src.txt
* Build.gradle에서 rootProject.name 값을 얻음. Single project는 적절히 지정
* 프로그램을 아래 예와 같이 실행 합니다. 멀티 프로젝트는 마지막에 ‘m’, 싱글 프로젝트는 ‘s’로 지정

python3 genprj.py src.txt myprj m

※ 소스 병합

* 아래 링크에서 mergesrc.py 파일을 c:\home\workspace 디렉토리에 다운로드

<https://drive.google.com/file/d/1LC3FRfeRELIbA3-kwl5fawJzMX1fMhH8/view?usp=drive_link>

* c:\home\workspace으로 이동
* python3 mergesrc.py 실행
* c:\home이 하위 디렉토리 중 합치고 싶은 디렉토리를 지정합니다.

예)

* python mergesrc.py:

‘cqrs’입력 : CQRS 프로젝트 전체 합침

* python mergesrc.py :   
  cqrs/command  
  CQRS의 command 서비스 소스만 합침
* python mergesrc.py :   
  cqrs/command cqrs/query  
  CQRS의 command, query 서비스 소스만 합침
* 합쳐진 결과 파일은 c:\home\workspace\MergedSource.java로 만들어집니다.

### 2) 프로그램 보완

1차 코딩 완료 후 InterlliJ에서 오픈한 후

실행을 위한 모든 코드가 개발되었는지 체크

| o: 실행을 위해 필요한 모든 클래스가 개발되었는지 체크해 주세요. |
| --- |

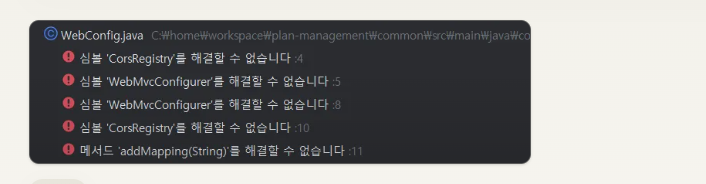
2차 코딩 완료 후 재 확인

| o: 다시 한번 누락된 클래스가 없는지 검토해 주세요. |
| --- |

### 3) 에러 Fix

IntelliJ에서 에러 텍스트를 복사하거나, 이미지 캡처하여 제공

반드시, 제일 상단에 있는 class명이 포함되게 캡처 해야 함



전체 소스를 자꾸 주려고 하면 아래와 같이 수정 필요한 소스만 달라고 요구

| 수정할 클래스만 보여주세요. |
| --- |

## 4. 프론트엔드 프로젝트 생성

### 1) 프로젝트 생성

클로드가 생성한 소스를 프로젝트 구조로 변환합니다.

* 아래 링크에서 genfront.py 파일을 c:\home\workspace디렉토리에 다운로드   
  <https://drive.google.com/file/d/1tmZKZ66BN8mbiMoSf_DMbf2Qfy5CIdk_/view?usp=drive_link>
* MobaXTerm에서 Local Ubuntu를 열고 홈으로 이동: cd 라고 치고 엔터 치면 됨
* 클로드가 제공한 소스를 한 파일로 합침: 예) front.txt
* 프로그램을 아래 예와 같이 실행 합니다.

python3 genfront.py front.txt myfront

※ 소스 병합

* 아래 링크에서 merge-node.py 파일을 c:\home\workspace 디렉토리에 다운로드  
  <https://drive.google.com/file/d/1UbE66R8Ht9fIVMSsVRaFCQK_c5x4ADT7/view?usp=drive_link>
* c:\home\workspace으로 이동
* python3 merge-node.py {frontend project}

예) python merge-node.py myfront

* 합쳐진 결과 파일은 c:\home\workspace\MergedFront.txt로 만들어집니다.

## 5. Database 설치(VM)

VM을 먼저 만드셔야 합니다. 아래 링크를 참고하세요.

VM작성

<https://github.com/cna-bootcamp/handson-azure/blob/main/prepare/setup-server.md#bastion-vm-%EC%83%9D%EC%84%B1>

MobaXTerm 세션 작성 및 접속

<https://github.com/cna-bootcamp/handson-azure/blob/main/prepare/setup-server.md#mobaxterm-%EC%84%B8%EC%85%98-%EC%9E%91%EC%84%B1>

### 1) Postgres

1. PostgreSQL 14 저장소 추가

| sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb\_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list' wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add - |
| --- |

2. 시스템 패키지 업데이트

| sudo apt update |
| --- |

3. PostgreSQL 14 설치

| sudo apt install -y postgresql-14 |
| --- |

4. PostgreSQL 서비스 상태 확인

| sudo systemctl status postgresql |
| --- |

결과 예시

| azureuser@bastion-ondal:~/stress$ sudo systemctl status postgresql ● postgresql.service - PostgreSQL RDBMS  Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)  Active: active (exited) since Tue 2025-01-07 10:11:52 KST; 11s ago  Main PID: 12198 (code=exited, status=0/SUCCESS)  CPU: 1ms  Jan 07 10:11:52 bastion-ondal systemd[1]: Starting PostgreSQL RDBMS... Jan 07 10:11:52 bastion-ondal systemd[1]: Finished PostgreSQL RDBMS. |
| --- |

5. PostgreSQL 설정 변경

| sudo vi /etc/postgresql/14/main/postgresql.conf |
| --- |

주석을 풀고 아래 설정 변경

| # 외부 접속을 위한 설정 변경 listen\_addresses = '\*' |
| --- |

6. 클라이언트 인증 설정

| sudo vi /etc/postgresql/14/main/pg\_hba.conf |
| --- |

맨 아래에 추가가

| # IPv4 원격 접속 허용 추가  host all all 0.0.0.0/0 md5 |
| --- |

7. PostgreSQL 서비스 재시작

| sudo systemctl restart postgresql |
| --- |

8. postgres 사용자 비밀번호 설정: 비밀번호는 복잡하게 작성

| sudo -u postgres psql postgres=# ALTER USER postgres PASSWORD 'Hi5Jessica!'; postgres=# \q |
| --- |

9. 정상설치 테스트

정상적으로 설치되었는지 테스트하려면 다음 명령어를 사용할 수 있습니다:

| psql -h localhost -U postgres -d postgres |
| --- |

10. 외부 접근 허용

- 포트 개방

sudo ufw allow 5432/tcp

- VM의 NSG(Network Security Group) 찾기

az network nsg list -o table

- PostgreSQL 포트(5432)에 대한 인바운드 규칙 추가

| az network nsg rule create \  --resource-group tiu-dgga-rg \  --nsg-name {NSG} \  --name Allow-PostgreSQL \  --priority 1001 \  --source-address-prefixes '\*' \  --source-port-ranges '\*' \  --destination-port-ranges 5432 \  --protocol Tcp \  --access Allow |
| --- |

예시)

| az network nsg rule create \  --resource-group tiu-dgga-rg \  --nsg-name bastion-ondalNSG \  --name Allow-PostgreSQL \  --priority 1001 \  --source-address-prefixes '\*' \  --source-port-ranges '\*' \  --destination-port-ranges 5432 \  --protocol Tcp \  --access Allow |
| --- |

10. 필요한 DB 생성

- PostgreSQL 서버에 접속

| sudo -u postgres psql |
| --- |

- 데이터베이스 생성

아래는 'telecomdb'생성 예시

| CREATE DATABASE telecomdb; |
| --- |

아래는 telecomdb를 user 'postgres'에 권한 부여하는 예시임

| GRANT ALL PRIVILEGES ON DATABASE telecomdb TO postgres; |
| --- |

- PostgreSQL 접속 종료

| \q |
| --- |

- 데이터베이스 생성 확인

| psql -h localhost -U postgres -d telecomdb |
| --- |

11. Dbeaver에서 접속

Database는 위에서

- VP IP구하기

| curl ifconfig.me |
| --- |

- DBeaver connection profile 생성

- Host: VM IP

- Port: 5432

- Database: postgres

- Username: postgres

- Password: Hi5Jessica!

\* Connection profile에서 Database 옆에 'Show all database' 체크

참고) Postgres 완전 삭제

| - PostgreSQL 서비스 중지  sudo systemctl stop postgresql  - PostgreSQL 패키지와 의존성 패키지 모두 제거  sudo apt-get remove --purge postgresql\*  - 설정 파일과 데이터 디렉토리 삭제  sudo rm -rf /etc/postgresql/  sudo rm -rf /var/lib/postgresql/  sudo rm -rf /var/log/postgresql/  - postgres 시스템 사용자 삭제  sudo userdel -r postgres  - 설치 저장소 제거  sudo rm /etc/apt/sources.list.d/pgdg.list  sudo rm /etc/apt/trusted.gpg.d/postgresql.gpg  - apt 캐시 정리  sudo apt update |
| --- |

### 2) Redis

Redis를 VM에 설치하는 상세한 방법을 안내해드리겠습니다:

1. Ubuntu 패키지 업데이트

| sudo apt update sudo apt upgrade |
| --- |

2. Redis 설치

| sudo apt install redis-server |
| --- |

3. Redis 설정파일 수정

| sudo vi /etc/redis/redis.conf |
| --- |

주요 설정 변경사항:

| # 외부 접속 허용을 위해 bind 주소 변경  bind 0.0.0.0  # 패스워드 설정 (보안을 위해 반드시 설정)  requirepass Hi5Jessica!  # 최대 메모리 설정 (시스템 메모리의 약 75% 권장)  maxmemory 1gb  # 메모리 정책 설정  maxmemory-policy allkeys-lru |
| --- |

4. Redis 서비스 재시작

| sudo systemctl restart redis-server |
| --- |

5. Redis 상태 확인

| sudo systemctl status redis-server |
| --- |

6. 외부 접근 허용 설정

- 방화벽 설정 (Redis 기본 포트 6379 오픈)

| sudo ufw allow 6379 |
| --- |

- VM의 NSG(Network Security Group) 찾기

az network nsg list -o table

- Redis 포트(6379)에 대한 인바운드 규칙 추가

{NSG}는 본인 것으로 변경해야 합니다.

| az network nsg rule create \  --resource-group tiu-dgga-rg \  --nsg-name {NSG} \  --name Allow-Redis \  --priority 1002 \  --source-address-prefixes '\*' \  --source-port-ranges '\*' \  --destination-port-ranges 6379 \  --protocol Tcp \  --access Allow |
| --- |

7. Redis 동작 테스트

| redis-cli  > auth your\_password  > set test "Hello Redis"  > get test |
| --- |

문제가 발생하면 로그를 확인하세요:

| sudo tail -f /var/log/redis/redis-server.log |
| --- |

## 6. Database 설치(k8s Pod)

사전에 아래 링크를 참조하여 **AKS, ACR**을 만드세요.

<https://github.com/cna-bootcamp/handson-azure/blob/main/prepare/setup-server.md#aksacr-%EC%83%9D%EC%84%B1-%EC%82%AD%EC%A0%9C>

주요 Database를 AKS에 설치하는 방법을 가이드 합니다.

Local Ubuntu 또는 Bastion VM에 로그인하여 작업하십시오.

Bitnami helm registry 추가

| $ helm repo ls # 없으면 추가 $ helm repo add bitnami https://charts.bitnami.com/bitnami $ helm repo update |
| --- |

작업 디렉토리 생성

| $ mkdir -p ~/install && cd ~/install |
| --- |

Namespace 생성 및 이동

| $ k create ns {team id}-ns $ kubens {team id}-ns |
| --- |

### 1) MongoDB

- 작업 디렉토리 생성

| $ mkdir -p ~/install/mongodb && cd ~/install/mongodb |
| --- |

- values.yaml 작성

실습 시에는 **architecture를 ‘standalone’으로 지정**해서 Pod를 1개만 생성하세요.

| # MongoDB 아키텍처 설정 architecture: replicaset replicaCount: 3 # 인증 설정 auth:  enabled: true  rootUser: root  rootPassword: "Hi5Jessica!"  database: "telecomdb"  username: "telecomuser"  password: "Hi5Jessica!"  replicaSetKey: "Hi5Jessica!"  # 리소스 설정 resources:  limits:  cpu: 1  memory: 2Gi  requests:  cpu: 0.5  memory: 1Gi # 스토리지 설정  persistence:  enabled: true  storageClass: "managed"  size: 10Gi # 성능 최적화 설정 extraEnvVars:  - name: MONGODB\_DISABLE\_SYSTEM\_LOG  value: "false"  - name: MONGODB\_ENABLE\_DIRECTORY\_PER\_DB  value: "true" # MongoDB 설정 최적화 extraFlags:  - "--wiredTigerCacheSizeGB=2"  - "--maxConns=100" # 네트워크 설정 service:  type: ClusterIP  ports:  mongodb: 27017 # Pod 안정성 설정 affinity:   podAntiAffinity:  preferredDuringSchedulingIgnoredDuringExecution:  - weight: 100  podAffinityTerm:  labelSelector:  matchLabels:  app.kubernetes.io/name: mongodb  app.kubernetes.io/instance: mongodb  topologyKey: kubernetes.io/hostname # 보안 설정 containerSecurityContext:  enabled: true  runAsUser: 1001  runAsNonRoot: true # 리소스 모니터링 livenessProbe:  enabled: true  initialDelaySeconds: 30  periodSeconds: 10  timeoutSeconds: 5  failureThreshold: 6  successThreshold: 1 readinessProbe:  enabled: true  initialDelaySeconds: 5  periodSeconds: 10  timeoutSeconds: 5  failureThreshold: 6  successThreshold: 1 |
| --- |

- 설치

| helm upgrade -i mongo -f values.yaml bitnami/mongodb --version 14.3.2 |
| --- |

아래 명령으로 모든 pod가 실행될때까지 기다렸다가, CTRL-C로 중지.

| watch kubectl get po |
| --- |

- 외부 접속을 위한 service 생성

mongo-external.yaml 작성

| apiVersion: v1  kind: Service  metadata:  name: mongo-external  spec:  type: LoadBalancer  ports:  - name: mongodb  port: 27017  targetPort: 27017  Selector:  app.kubernetes.io/component: mongodb  app.kubernetes.io/name: mongodb |
| --- |

Service 객체 생성

| k apply -f mongo-external.yaml |
| --- |

L/B IP 확인

| k get svc |
| --- |

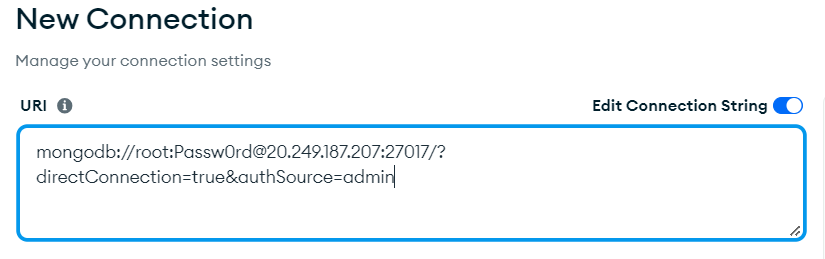
- 로컬에서 접속 테스트

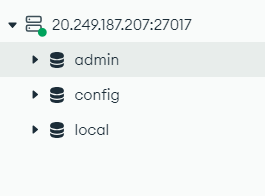
아래 사이트에서 Compass 설치

<https://www.mongodb.com/try/download/compass>

실행 후 아래와 같이 연결 문자열 입력하여 연결. **IP는 위 service의 L/B IP로 바꿔야 함**

| mongodb://root:Hi5Jessica!@20.249.187.207:27017/?directConnection=true&authSource=admin |
| --- |





### 2) Postgres

- 작업 디렉토리 생성

| $ mkdir -p ~/install/postgres && cd ~/install/postgres |
| --- |

- values.yaml 작성

실습 시에는 **architecture를 ‘standalone’으로 지정**해서 Pod를 1개만 생성하세요.

| # PostgreSQL 아키텍처 설정  architecture: replication  # 글로벌 설정  global:  postgresql:  auth:  postgresPassword: "Hi5Jessica!"  replicationPassword: "Hi5Jessica!"  database: "telecomdb"  username: "telecomuser"  password: "Hi5Jessica!"  storageClass: "managed-premium"    # Primary 설정  primary:  persistence:  enabled: true  storageClass: "managed-premium"  size: 10Gi    resources:  limits:  memory: "4Gi"  cpu: "1"  requests:  memory: "2Gi"  cpu: "0.5"    # 성능 최적화 설정  extraEnvVars:  - name: POSTGRESQL\_SHARED\_BUFFERS  value: "1GB"  - name: POSTGRESQL\_EFFECTIVE\_CACHE\_SIZE  value: "3GB"  - name: POSTGRESQL\_MAX\_CONNECTIONS  value: "200"  - name: POSTGRESQL\_WORK\_MEM  value: "16MB"  - name: POSTGRESQL\_MAINTENANCE\_WORK\_MEM  value: "256MB"  # 고가용성 설정  podAntiAffinityPreset: soft    # Secondary 설정  readReplicas:  replicaCount: 2    persistence:  enabled: true  storageClass: "managed-premium"  size: 10Gi  resources:  limits:  memory: "2Gi"  cpu: "1"  requests:  memory: "1Gi"  cpu: "0.5"  # 성능 최적화 설정  extraEnvVars:  - name: POSTGRESQL\_SHARED\_BUFFERS  value: "1GB"  - name: POSTGRESQL\_EFFECTIVE\_CACHE\_SIZE  value: "3GB"  - name: POSTGRESQL\_MAX\_CONNECTIONS  value: "200"  - name: POSTGRESQL\_WORK\_MEM  value: "16MB"  - name: POSTGRESQL\_MAINTENANCE\_WORK\_MEM  value: "256MB"  # 고가용성 설정  podAntiAffinityPreset: soft  # 네트워크 설정  service:  type: ClusterIP  ports:  postgresql: 5432  # 보안 설정  securityContext:  enabled: true  fsGroup: 1001  runAsUser: 1001 |
| --- |

- namespace 생성(필요시)

| k create ns {namespace}  kubens {namespace} |
| --- |

- 설치

| helm upgrade -i postgres -f values.yaml bitnami/postgresql --version 14.3.2 |
| --- |

아래 명령으로 모든 pod가 실행될때까지 기다렸다가, CTRL-C로 중지.

| watch kubectl get po |
| --- |

- 외부 접속을 위한 service 생성

postgres-external.yaml 작성

| apiVersion: v1  kind: Service  metadata:  name: postgres-external  spec:  ports:  - name: tcp-postgresql  port: 5432  protocol: TCP  targetPort: tcp-postgresql  selector:  app.kubernetes.io/component: primary  app.kubernetes.io/instance: postgres  app.kubernetes.io/name: postgresql  sessionAffinity: None  type: LoadBalancer |
| --- |

Service 객체 생성

| k apply -f postgres-external.yaml |
| --- |

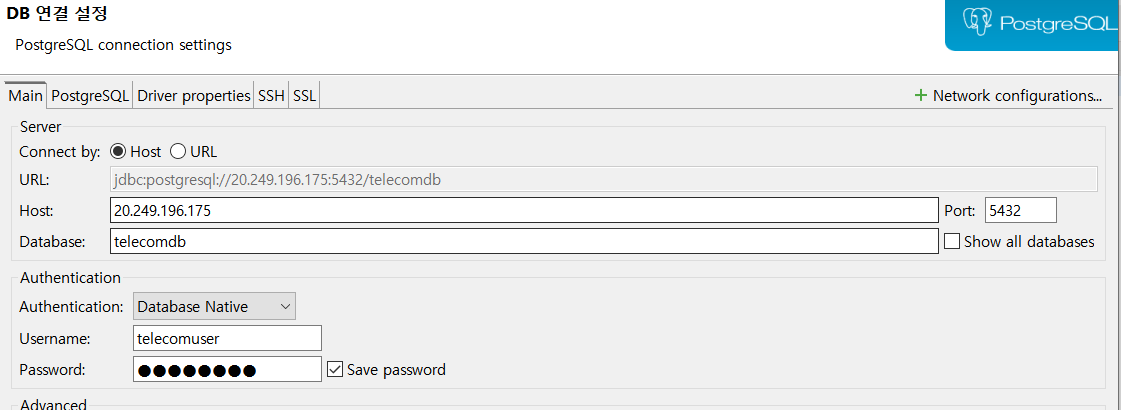
L/B IP 확인

| k get svc |
| --- |

- 로컬에서 접속 테스트

DBeaver를 실행하고 Postgres용 연결 설정 파일을 만들어 테스트 합니다.

Host는 위 service 객체의 L/B IP이고, id와 pw는 values.yaml에 지정한 값을 사용합니다.



### 3) Redis

- 작업 디렉토리 생성

| $ mkdir -p ~/install/redis && cd ~/install/redis |
| --- |

- values.yaml 작성

실습 시에는 **architecture를 ‘standalone’으로 지정**해서 Pod를 1개만 생성하세요.

| architecture: replication  auth:  enabled: true  password: "Hi5Jessica!"  master:  persistence:  enabled: true  storageClass: "managed"  size: 10Gi  configuration: |  maxmemory 1610612736  maxmemory-policy allkeys-lru  appendonly yes  appendfsync everysec  save 900 1 300 10 60 10000    resources:  limits:  memory: "2Gi"  cpu: "1"  requests:  memory: "1Gi"  cpu: "0.5"    replica:  replicaCount: 2  persistence:  enabled: true  storageClass: "managed"  size: 10Gi  configuration: |  maxmemory 1610612736  maxmemory-policy allkeys-lru  resources:  limits:  memory: "2Gi"  cpu: "1"  requests:  memory: "1Gi"  cpu: "0.5"  sentinel:  enabled: true  quorum: 2    service:  type: ClusterIP  ports:  redis: 6379  podAntiAffinityPreset: soft  securityContext:  enabled: true  fsGroup: 1001  runAsUser: 1001 |
| --- |

- 설치

| helm upgrade -i redis -f values.yaml bitnami/redis --version 18.4.0 |
| --- |

아래 명령으로 모든 pod가 실행될때까지 기다렸다가, CTRL-C로 중지.

| watch kubectl get po |
| --- |

- 외부 접속을 위한 service 생성

redis-external.yaml 작성

| apiVersion: v1  kind: Service  metadata:  name: redis-external  spec:  ports:  - name: tcp-redis  port: 6379  protocol: TCP  targetPort: redis  - name: tcp-sentinel  port: 26379  protocol: TCP  targetPort: redis-sentinel  publishNotReadyAddresses: true  selector:  app.kubernetes.io/instance: redis  app.kubernetes.io/name: redis  sessionAffinity: None  type: LoadBalancer |
| --- |

Service 객체 생성

| k apply -f redis-external.yaml |
| --- |

L/B IP 확인

| k get svc |
| --- |

- 로컬에서 접속 테스트

아래 사이트에서 redis insight를 다운로드하여 설치합니다.

<https://redis.io/insight/>

아래와 같이 셋팅하고 연결 테스트를 합니다.

IP는 위에서 생성한 L/B IP이고, 암호는 values.yaml에 지정한 값입니다.

| redis://:Password@20.249.200.253:6379 |
| --- |

### 

## 7. MQ설치

**※ Azure MQ 서비스 정보**

| **비교 요소** | **Service Bus** | **Event Hubs(Kafka기반)** | **Event Grid** |
| --- | --- | --- | --- |
| **주요 용도** | 엔터프라이즈 메시징 | 빅데이터 스트리밍 | 이벤트 라우팅 |
| **Consumer 수** | **1:1 또는 경쟁 소비자** | **파티션당 1개 Consumer Group** | **1:N(무제한)** |
| **최대 메시지 크기** | 100MB (Premium) | 1MB | 1MB |
| **순서 보장** | 완벽한 FIFO | 파티션 내 보장 | 미보장 |
| **중복 전달** | Exactly-once  중복 전달 없음 | At-least-once  중복 전달 가능 | At-least-once  중복 전달 가능 |
| **처리량** | 중 | 상 | 중 |
| **지연 시간** | 낮음 (~ms) | 매우 낮음 (< ms) | 매우 낮음 (< ms) |
| **메시지 필터링** | 지원 | 미지원 | 지원 |
| **부하 분산** | 내장 | 파티션 기반 | 자동 |
| **상대적 비용** | 상 | 중 | 하 |
| **적합 사례** | • 금융 거래  • 주문 처리  • 재고 관리 | • IoT 데이터  • 실시간 분석  • 로그 수집 | • **이벤트 브로드캐스팅(1:N)**  • 서버리스 아키텍처  • 서비스 통합 |
| **부적합 사례** | • 대용량 스트리밍  • 브로드캐스팅  • 실시간 모니터링 | • 정확한 순서 필요  • 개별 메시지 처리  • 메시지 필터링 | • 대용량 스트리밍  • 순서 보장 필요  • 장기 재시도 필요 |

**적합/부적합 사례**

| **MQ** | **적합** | **부적합** |
| --- | --- | --- |
| **Service Bus** | ✓ 순서가 중요하고 중복되면 안 되는 경우  (예: 은행 입출금)  ✓ 안전하게 처리해야 하는 경우  (예: 쇼핑몰 주문/결제)  ✓ 처리상태를 추적해야 하는 경우  (예: 택배 배송상태)  ✓ 실패하면 다시 시도해야 하는 중요한 업무 | ✗ 실시간 영상/데이터를 처리하는 경우  (예: 유튜브)  ✗ 여러 사람에게 동시에 보내야 하는 경우  (예: 카톡)  ✗ 초당 수천 건 이상 처리해야 하는 경우 |
| **Event Hub** | ✓ 스마트폰/IoT 기기의 실시간 데이터 수집  ✓ 서버/앱의 **로그**를 **실시간**으로 **수집**하는 경우  ✓ 실시간 차트/그래프를 그려야 하는 경우  ✓ 초당 수천~수만 건을 빠르게 처리해야 하는 경우 | ✗ 송금처럼 순서가 중요한 거래  ✗ 택배처럼 개별 메시지를 추적해야 하는 경우  ✗ 특정 메시지만 골라서 처리해야 하는 경우 |
| **Event Grid** | ✓ 여러 시스템에 동시에 알림/푸시  ✓ 마이크로서비스 간 1:N 이벤트 전달 | ✗ 대용량 파일/데이터 전송  ✗ 순서대로 처리해야 하는 경우  ✗ 실패한 메시지를 다시 처리해야 하는 경우  ✗ 복잡한 조건으로 메시지를 분류 |

**사전준비**

* TEAM ID 정하기
* Resource Group 확인: az group list -o table로 확인
* Location: 위 리소스 그룹 확인 시 같이 확인

### 1) Service Bus

TEAMID, Topic명(usage, notify)을 적절히 변경.

| export TEAMID=unicorn  export RESOURCE\_GROUP="tiu-dgga-rg" export LOCATION="koreacentral" export SB\_NAMESPACE="sb-${NAME}"  export DEADLETTER\_TOPIC="deadletter-topic"  export SB\_TOPIC1="usage" export SB\_TOPIC2="notify"  export SB\_SUB1="sync-sub"  export SB\_SUB2="notification-sub"  export MAX\_RETRY=3  # Namespace 생성 if ! az servicebus namespace show --name $SB\_NAMESPACE -g $RESOURCE\_GROUP &>/dev/null; then  az servicebus namespace create \  --name $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP \  --location $LOCATION \  --sku Standard  check\_error "Service Bus Namespace 생성 실패" fi  # Topic 생성 for topic in $SB\_TOPIC1 $SB\_TOPIC2; do  if ! az servicebus topic show --name $topic --namespace $SB\_NAMESPACE -g $RESOURCE\_GROUP &>/dev/null; then  az servicebus topic create \  --name $topic \  --namespace $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP  check\_error "Topic $topic 생성 실패"  fi done  # Dead Letter Queue용 Topic 생성  az servicebus topic create \  --name $DEADLETTER\_TOPIC \  --namespace $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP  # Subscription 생성 az servicebus topic subscription create \  --name $SB\_SUB1 \  --namespace $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP \  --topic-name $SB\_TOPIC1  --forward-deadlettered-messages-to $DEADLETTER\_TOPIC \  --enable-dead-lettering-on-message-expiration true \  --max-delivery-count $MAX\_RETRY  az servicebus topic subscription create \  --name $SB\_SUB2 \  --namespace $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP \  --topic-name $SB\_TOPIC1  --forward-deadlettered-messages-to $DEADLETTER\_TOPIC \  --enable-dead-lettering-on-message-expiration true \  --max-delivery-count $MAX\_RETRY  # Connection String 가져오기 SB\_CONNECTION\_STRING=$(az servicebus namespace authorization-rule keys list \  --name RootManageSharedAccessKey \  --namespace $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP \  --query primaryConnectionString -o tsv)  # display result echo "connection string => $SB\_CONNECTION\_STRING" echo "Topic1 => $SB\_TOPIC1"  echo "Topic2 => $SB\_TOPIC2"  echo "DEAD LETTER Topic => $DEADLETTER\_TOPIC"  echo "Sub1 => $SB\_SUB1"  echo "Sub22 => $SB\_SUB2" |
| --- |

※ 생성된 객체 삭제

| # Subscription 삭제  az servicebus topic subscription delete \  --name $SB\_SUB1 \  --namespace-name $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP \  --topic-name $SB\_TOPIC1 \  2>/dev/null || true  az servicebus topic subscription delete \  --name $SB\_SUB2 \  --namespace-name $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP \  --topic-name $SB\_TOPIC2 \  2>/dev/null || true  # Topic 삭제  az servicebus topic delete \  --name $SB\_TOPIC1 \  --namespace-name $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP \  2>/dev/null || true  az servicebus topic delete \  --name $SB\_TOPIC2 \  --namespace-name $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP \  2>/dev/null || true  az servicebus topic delete \  --name $DEADLETTER\_TOPIC \  --namespace-name $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP \  2>/dev/null || true  # Namespace 삭제  az servicebus namespace delete \  --name $SB\_NAMESPACE \  --resource-group $RESOURCE\_GROUP \  2>/dev/null || true |
| --- |

### 2) Event Hub

TEAMID, STORAGEID, EVENTHUB\_NS, EVENTHUB\_NAME을 적절히 변경.

STORAGEID은 전세계 Azure cloud에서 유일해야 함.

| export TEAMID=unicorn  export ROOT\_PROJECT=pubsub export RESOURCE\_GROUP=tiu-dgga-rg export LOCATION=koreacentral export STORAGEID=${TEAMID}storage${ROOT\_PROJECT} export EVENTHUB\_NS=${TEAMID}-ns-${ROOT\_PROJECT} export EVENTHUB\_NAME=${TEAMID}-name-${ROOT\_PROJECT} export BLOB\_CONTAINER=${TEAMID}-checkpoints-${ROOT\_PROJECT}  # Storage Account 존재 체크  az storage account show \  --name $STORAGEID \  --resource-group $RESOURCE\_GROUP \  --query name \  --output tsv  # Storage Account 없으면 생성  az storage account create \  --name ${STORAGEID} \  --resource-group ${RESOURCE\_GROUP} \  --location ${LOCATION} \  --sku Standard\_LRS \  --only-show-errors  # Get Storage connection string STORAGE\_CONNECTION=$(az storage account show-connection-string \ --name ${STORAGEID} \ --resource-group ${RESOURCE\_GROUP} \ --query connectionString \ --output tsv)  # Create container az storage container create \ --name ${BLOB\_CONTAINER} \ --connection-string "${STORAGE\_CONNECTION}"  # Event Hubs setup az eventhubs namespace create \ --name ${EVENTHUB\_NS} \ --resource-group ${RESOURCE\_GROUP} \ --location ${LOCATION} \ --sku Basic  #EventHub 생성: partition수는 Pod수와 동일하게 함 az eventhubs eventhub create \ --name ${EVENTHUB\_NAME} \ --namespace-name ${EVENTHUB\_NS} \ --resource-group ${RESOURCE\_GROUP} \ --partition-count 1 \ --cleanup-policy Delete \ --retention-time 24  # Get connection strings EVENTHUB\_CONNECTION=$(az eventhubs namespace authorization-rule keys list \ --resource-group ${RESOURCE\_GROUP} \ --namespace-name ${EVENTHUB\_NS} \ --name RootManageSharedAccessKey \ --query primaryConnectionString -o tsv)  # Save connection strings to files echo "STORAGE\_CONNECTION => $STORAGE\_CONNECTION" echo "EVENTHUB\_CONNECTION => $EVENTHUB\_CONNECTION" |
| --- |

※ 생성된 객체 삭제

| STORAGE\_CONNECTION=$(az storage account show-connection-string \ --name ${STORAGEID} \ --resource-group ${RESOURCE\_GROUP} \ --query connectionString \ --output tsv)  # Event Hub 삭제 az eventhubs eventhub delete \  --name ${EVENTHUB\_NAME} \  --namespace-name ${EVENTHUB\_NS} \  --resource-group ${RESOURCE\_GROUP}  # Event Hub Namespace 삭제 az eventhubs namespace delete \  --name ${EVENTHUB\_NS} \  --resource-group ${RESOURCE\_GROUP}  # Blob Container 삭제  az storage container delete \  --name ${BLOB\_CONTAINER} \  --connection-string "${STORAGE\_CONNECTION}"  # Storage Account 삭제 az storage account delete \  --name ${STORAGEID} \  --resource-group ${RESOURCE\_GROUP} \  --yes |
| --- |

### 3) EventGrid

- 작업 디렉토리 생성

| $ mkdir -p ~/install/eventgrid && cd ~/install/eventgrid |
| --- |

- Domain추가하기

아래 내용으로 shell 생성. 파일명은 domain.sh

| #!/bin/bash  if [ $# -ne 1 ]; then  echo "Usage: $0 TEAMID"  echo "Example: $0 unicorn"  exit 1  fi  TEAMID=$1  DOMAIN\_SUFFIX="4.217.249.140.nip.io"  DOMAIN="${TEAMID}.${DOMAIN\_SUFFIX}"  NGINX\_SITES\_AVAILABLE="/etc/nginx/sites-available"  NGINX\_SITES\_ENABLED="/etc/nginx/sites-enabled"  echo "Creating certificate for ${DOMAIN}..."  sudo certbot --nginx -d $DOMAIN  CONFIG\_FILE="${NGINX\_SITES\_AVAILABLE}/${TEAMID}"  echo "Creating Nginx configuration for ${DOMAIN}..."  sudo bash -c "cat > ${CONFIG\_FILE}" <<'EOL'  server {  listen 443 ssl;  server\_name DOMAIN\_PLACEHOLDER;  ssl\_certificate /etc/letsencrypt/live/DOMAIN\_PLACEHOLDER/fullchain.pem;  ssl\_certificate\_key /etc/letsencrypt/live/DOMAIN\_PLACEHOLDER/privkey.pem;  ssl\_protocols TLSv1.2 TLSv1.3;  ssl\_ciphers HIGH:!aNULL:!MD5;  location / {  #proxy\_pass http://20.214.113.85:80;  proxy\_ssl\_verify off;  proxy\_buffer\_size 64k;  proxy\_buffers 4 64k;  proxy\_busy\_buffers\_size 64k;  proxy\_set\_header Host $host;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header X-Forwarded-Proto $scheme;  proxy\_read\_timeout 60s;  proxy\_connect\_timeout 60s;  proxy\_send\_timeout 60s;  }  }  EOL  sudo sed -i "s/DOMAIN\_PLACEHOLDER/${DOMAIN}/g" "${CONFIG\_FILE}"  sudo ln -sf ${CONFIG\_FILE} ${NGINX\_SITES\_ENABLED}/${TEAMID}  echo "Reloading Nginx..."  sudo systemctl reload nginx  echo "Configuration complete for ${DOMAIN}" |
| --- |

실행파일로 전환

| chmod +x domain.sh |
| --- |

실행하여 도메인 {Team id}.4.217.249.140.nip.io 생성

| ./domain.sh unicorn |
| --- |

웹브라우저에서 https연결 확인

https://unicorn.4.217.249.140.nip.io/

- Event Grid Topic, Dead Letter 생성

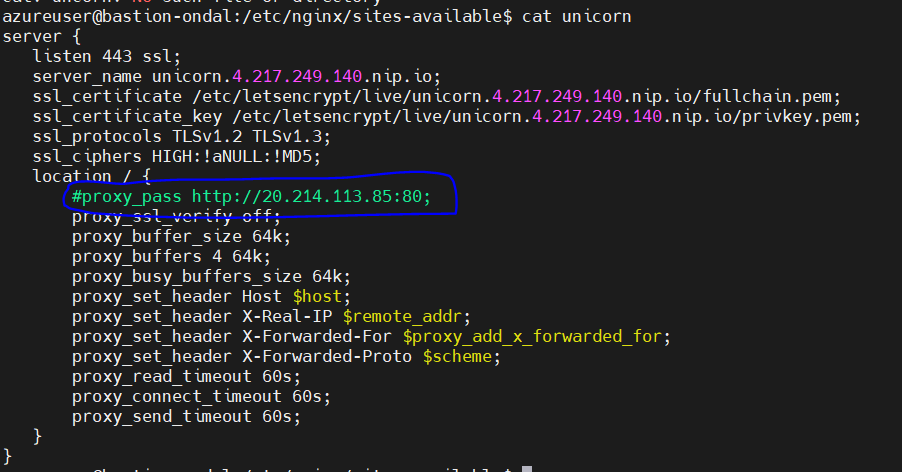
TEAMID, STORAGEID, EG\_TOPIC, EG\_SUB, DEAD\_LETTER는 적절히 수정

STORAGEID은 전세계 Azure cloud에서 유일해야 함..

| export TEAMID=unicorn  export ROOT\_PROJECT=pubsub export RESOURCE\_GROUP=tiu-dgga-rg export LOCATION=koreacentral export STORAGEID=${TEAMID}storage${ROOT\_PROJECT}  export EG\_TOPIC="${TEAMID}-topic-${ROOT\_PROJECT}"  export EG\_SUB="${TEAMID}-sub-${ROOT\_PROJECT}"  export DEAD\_LETTER="${TEAMID}deadletter${ROOT\_PROJECT}"  export PROXY\_IP="4.217.249.140"  # Event Grid Topic 생성  az eventgrid topic create \  --name $EG\_TOPIC \  --resource-group $RESOURCE\_GROUP \  --location $LOCATION \  --output none  # Storage Account가 있는지 체크  az storage account show \  --name $STORAGEID \  --resource-group $RESOURCE\_GROUP \  --query name \  --output tsv  # Storage Account 없으면 생성  az storage account create \  --name $STORAGEID \  --resource-group $RESOURCE\_GROUP \  --location $LOCATION \  --sku Standard\_LRS  # Get Storage connection string  STORAGE\_CONNECTION=$(az storage account show-connection-string \  --name ${STORAGEID} \  --resource-group ${RESOURCE\_GROUP} \  --query connectionString \  --output tsv)  # deadletter 컨테이너 존재 여부 확인  az storage container exists \  --name $DEAD\_LETTER \  --connection-string "${STORAGE\_CONNECTION}" \  --query "exists" \  -o tsv  # deadletter 컨테이너 없으면 생성  az storage container create \  --name $DEAD\_LETTER \  --connection-string "${STORAGE\_CONNECTION}" \  --output none |
| --- |

- Event Grid Subscriber 설정  
Nginx proxy에서 proxy\_pass를 Subscriber 애플리케이션으로 변경해야 함

| sudo vi /etc/nginx/sites-available/{team id} |
| --- |



변경후 nginx 재시작

| sudo nginx -t sudo systemctl restart nginx |
| --- |

- Event Grid Subscriber 등록

SUB\_ENDPOINT를 맞게 수정하고 실행

| # Event Grid Subscriber 설정 export SUB\_ENDPOINT="https://${TEAMID}.${PROXY\_IP}.nip.io/api/…"  # 기존 subscription 확인 az eventgrid event-subscription show \ --name $EG\_SUB \ --source-resource-id $(az eventgrid topic show --name $EG\_TOPIC -g $RESOURCE\_GROUP --query "id" -o tsv) \ --query "provisioningState" -o tsv  # subscription 없으면 생성 storage\_id=$(az storage account show \  --name $STORAGEID \  --resource-group $RESOURCE\_GROUP \  --query id \  --output tsv)   az eventgrid event-subscription create \  --name $EG\_SUB \  --source-resource-id $(az eventgrid topic show --name $EG\_TOPIC -g $RESOURCE\_GROUP --query "id" -o tsv) \  --endpoint $SUB\_ENDPOINT \  --endpoint-type webhook \  --included-event-types UsageExceeded UsageAlert \  --max-delivery-attempts 3 \  --event-ttl 1440 \  --deadletter-endpoint "${storage\_id}/blobServices/default/containers/${DEAD\_LETTER}" \  --output none |
| --- |

※ EventGrid관련 객체 삭제

| # Event Grid Subscription 삭제 az eventgrid event-subscription delete \  --name $EG\_SUB \  --source-resource-id $(az eventgrid topic show --name $EG\_TOPIC -g $RESOURCE\_GROUP --query "id" -o tsv)  # Event Grid Topic 삭제  az eventgrid topic delete \  --name $EG\_TOPIC \  --resource-group $RESOURCE\_GROUP  # Blob Container 삭제 az storage container delete \  --name $DEAD\_LETTER \  --connection-string "${STORAGE\_CONNECTION}"  # Storage Account 삭제 az storage account delete \  --name $STORAGEID \  --resource-group $RESOURCE\_GROUP \  --yes |
| --- |

### 4) RabbitMQ

- 작업 디렉토리 생성

| $ mkdir -p ~/install/rabbitmq && cd ~/install/rabbitmq |
| --- |

- manifest 생성

deploy.yaml 로 생성.

| apiVersion: apps/v1  kind: StatefulSet  metadata:  name: rabbitmq  labels:  app: rabbitmq  spec:  serviceName: rabbitmq  selector:  matchLabels:  app: rabbitmq  replicas: 1  template:  metadata:  name: rabbitmq  labels:  app: rabbitmq  spec:  serviceAccountName: default  containers:  - name: rabbitmq  image: rabbitmq:management  imagePullPolicy: IfNotPresent  env:  - name: RABBITMQ\_DEFAULT\_USER  value: admin  - name: RABBITMQ\_DEFAULT\_PASS  value: P@ssw0rd$  ports:  - name: containerport  containerPort: 5672  - name: consoleport  containerPort: 15672  resources:  requests:  cpu: 128m  memory: 128Mi  limits:  cpu: 1024m  memory: 1024Mi  ---  apiVersion: v1  kind: Service  metadata:  name: rabbitmq  spec:  type: LoadBalancer  selector:  app: rabbitmq  ports:  - name: port1  port: 5672  targetPort: 5672  - name: port2  port: 15672  targetPort: 15672 |
| --- |

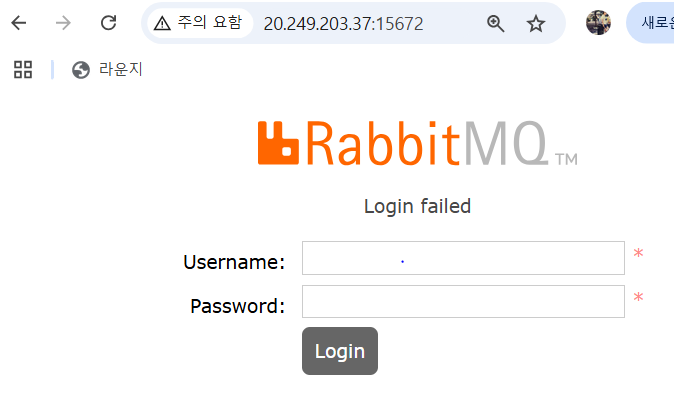
- 설치

| k apply -f deploy.yaml |
| --- |

- Admin 페이지 접속

L/B IP 확인하여 [http://{rabbit](about:blank) mq service L/B IP}:15672로 접속.

ID/PW는 deploy.yaml에 지정한 값으로 로그인.



### 

### 

### 5) Kafka

- 작업 디렉토리 생성

| $ mkdir -p ~/install/kafka && cd ~/install/kafka |
| --- |

- values.yaml 작성

| # values.yaml  global:  storageClass: "managed"  auth:  clientProtocol: plaintext  interBrokerProtocol: plaintext  sasl:  enabled: false  tls:  enabled: false  listeners:  client:  protocol: PLAINTEXT    kraft:  enabled: true  controller:  replicaCount: 1  heapOpts: "-Xmx1g -Xms1g"  persistence:  enabled: true  size: 10Gi  resources:  limits:  memory: 2Gi  cpu: 1  requests:  memory: 1Gi  cpu: 1  broker:  replicaCount: 3  heapOpts: "-Xmx1g -Xms1g"  persistence:  enabled: true  size: 10Gi  resources:  limits:  memory: 2Gi  cpu: 1  requests:  memory: 1Gi  cpu: 1  deleteTopicEnable: true  autoCreateTopicsEnable: false  offsets:  topic:  replication:  factor: 3  segment:  bytes: 1073741824 |
| --- |

- 설치

| helm upgrade -i kafka -f values.yaml bitnami/kafka --version 29.3.14 |
| --- |

아래 명령으로 모든 pod가 실행될때까지 기다렸다가, CTRL-C로 중지.

| watch kubectl get po |
| --- |

- Kafka web console 파드를 실행

kafka-ui.yaml 작성

| apiVersion: apps/v1  kind: Deployment  metadata:  name: kafka-ui  labels:  app: kafka-ui  spec:  replicas: 1  selector:  matchLabels:  app: kafka-ui  template:  metadata:  labels:  app: kafka-ui  spec:  containers:  - name: kafka-ui  image: provectuslabs/kafka-ui:latest  env:  - name: KAFKA\_CLUSTERS\_0\_NAME  value: "k8s-kafka"  - name: KAFKA\_CLUSTERS\_0\_BOOTSTRAPSERVERS  value: "kafka:9092"  - name: KAFKA\_CLUSTERS\_0\_PROPERTIES\_SECURITY\_PROTOCOL  value: "PLAINTEXT"  - name: KAFKA\_CLUSTERS\_0\_PROPERTIES\_SASL\_ENABLED  value: "false"  ports:  - containerPort: 8080  name: http  resources:  limits:  memory: "1Gi"  cpu: "500m"  requests:  memory: "512Mi"  cpu: "250m"  ---  apiVersion: v1  kind: Service  metadata:  name: kafka-ui  labels:  app: kafka-ui  spec:  type: LoadBalancer  ports:  - port: 8080  targetPort: http  protocol: TCP  name: http  selector:  app: kafka-ui |
| --- |

객체 생성

| k apply -f kafka-ui.yaml |
| --- |

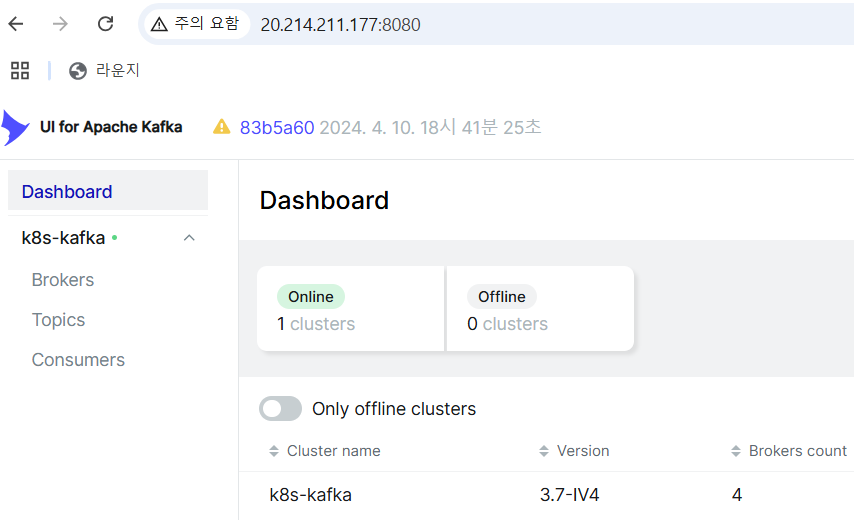
L/B IP 확인

| k get svc kafka-ui |
| --- |

- 접속 테스트

웹브라우저에서 web console 접근.

[http://{kafka-ui service L/B IP}:8080](about:blank)



### 

## 8. Azure Blob Storage 설치

RESOURCE\_GROUP, LOCATION을 확인하고 틀리면 바꾸세요.

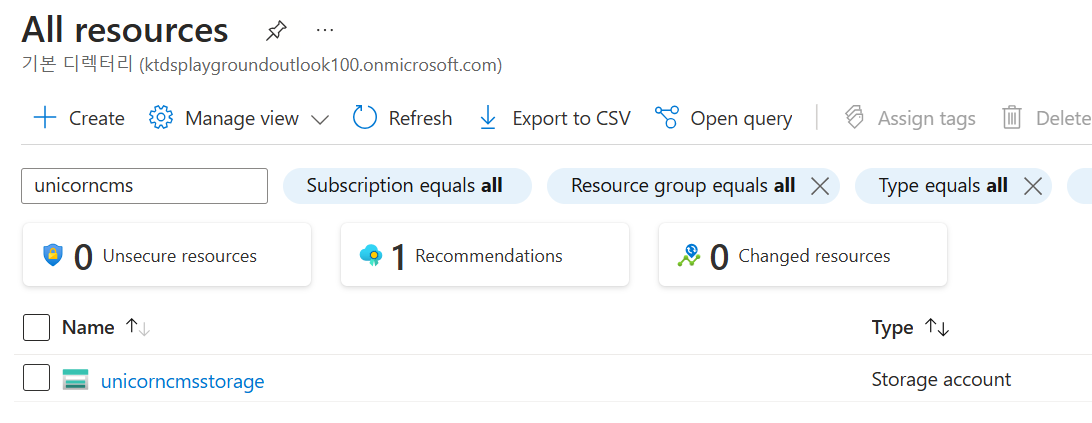
| az group list -o table |
| --- |

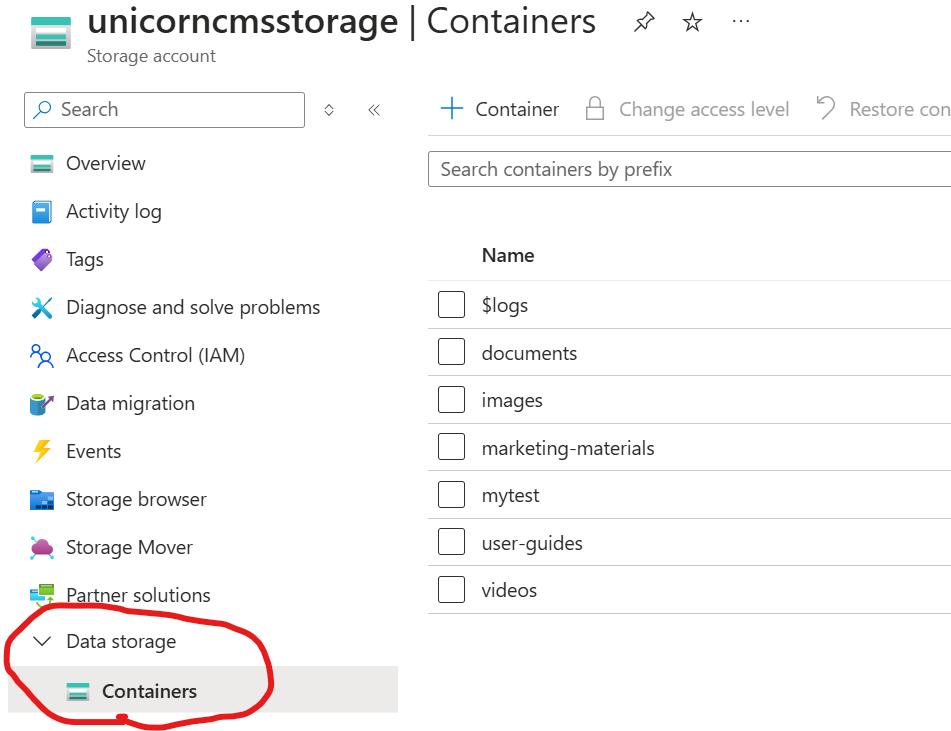
USERID, SYSTEM, CONTAINERS를 본인 서비스에 맞게 수정합니다.

| #!/bin/bash  RESOURCE\_GROUP="ictcoe-edu" LOCATION="koreacentral"  USERID="unicorn"  SYSTEM="cms" NAME="${USERID}-${SYSTEM}" STORAGE\_ACCOUNT\_NAME="${USERID}${SYSTEM}storage"  # 생성할 컨테이너 목록 정의 CONTAINERS=("marketing-materials" "user-guides" "documents" "images" "videos")  # 로그 함수 정의 log() {  echo "[INFO] $1" }  # 에러 체크 함수 정의 check\_error() {  if [ $? -ne 0 ]; then  echo "[ERROR] $1"  exit 1  fi }   # Storage 계정 존재 확인 log "Storage 계정 확인 중..." az storage account show --name $STORAGE\_ACCOUNT\_NAME --resource-group $RESOURCE\_GROUP &>/dev/null if [ $? -ne 0 ]; then  log "Storage 계정 생성 중..."  az storage account create \  --name $STORAGE\_ACCOUNT\_NAME \  --resource-group $RESOURCE\_GROUP \  --location $LOCATION \  --sku Standard\_LRS \  --kind StorageV2  check\_error "Storage 계정 생성 실패" else  log "기존 Storage 계정 사용" fi  # Azure CLI를 사용하여 CORS 설정 log "CORS 설정 중..." az storage cors clear --account-name $STORAGE\_ACCOUNT\_NAME --services b check\_error "CORS 초기화 실패"  # 새 CORS 규칙 추가 az storage cors add --account-name $STORAGE\_ACCOUNT\_NAME \  --services b \  --origins "\*" \  --methods "GET,HEAD,PUT,POST,DELETE,OPTIONS" \  --allowed-headers "\*" \  --exposed-headers "\*" \  --max-age 3600 check\_error "CORS 규칙 추가 실패"  # 연결 문자열 가져오기 log "Storage 연결 문자열 가져오는 중..." STORAGE\_CONNECTION\_STRING=$(az storage account show-connection-string \  --name $STORAGE\_ACCOUNT\_NAME \  --resource-group $RESOURCE\_GROUP \  --query connectionString \  --output tsv)   if [ $? -ne 0 ]; then  echo "[ERROR] Storage 연결 문자열 가져오기 실패"  exit 1 fi  # 지정된 컨테이너 생성 log "Blob Storage 컨테이너 생성 중..." for CONTAINER in "${CONTAINERS[@]}"; do  log "컨테이너 '$CONTAINER' 생성 중..."  az storage container create \  --name "$CONTAINER" \  --connection-string "$STORAGE\_CONNECTION\_STRING" \  --public-access off 2>/dev/null || true    if [ $? -eq 0 ]; then  log "컨테이너 '$CONTAINER' 생성 완료"  else  log "컨테이너 '$CONTAINER'가 이미 존재하거나 생성 중 오류 발생"  fi done  log "Azure Storage 설정 완료" |
| --- |

아래와 같이 Azure 포탈에서 확인합니다.

STORAGE\_ACCOUNT\_NAME에 해당하는 Storage Account 객체 찾기





위 컨테이너에 수동으로 파일을 올릴 수 있습니다.

# 테스트 코드 작성

## 1) Instruction에 추가

| [테스트 코드 작성]  {공통 요청사항}  모든 테스트 코드 시 공통으로 적용되는 요청사항입니다.  - 'TDD이해'의 '공통전략'을 준용하여 개발  - 네이밍룰은 'TDD이해'에 정의된 패키지, 클래스, 메소드, 테스트 데이터 네이밍룰을 준용  - '개발주석표준'에 따라 주석을 작성.  - 테스트 코드 개발 툴은 'Mockito'를 사용  - SprintBoot 3.4.0이후 버전에 맞는 Mockito 라이브러리 사용  - JSON으로 직렬화 되었다가 다시 역직렬화 되어 새로운 객체가 생기는 경우 'any' ArgumentMatcher 사용  {단위 테스트 코드 작성}  'tu:'로 시작하면 단위테스트 코드 작성 요청임  <요청사항>  - [테스트 코드 작성]의 {공통 요청사항}을 준용하여 개발  - 'TDD이해'의 '단위 테스트 전략'을 준용하여 개발  - 요청된 {서비스}의 소스를 분석하여 단위 테스트가 필요한 클래스와 Public Method를 식별  <참고자료>  - TDD이해  - MergedSource  <응답순서>  - 테스트 코드 패키지와 클래스 구조도  - 테스트 클래스 선정 이유  - 테스트 코드: 각 클래스마다 별도의 코드블록으로 제공  <응답형식>  코드블록  {통합 테스트 코드 작성}  'ti:'로 시작하면 통합테스트 코드 작성 요청임  <요청사항>  - [테스트 코드 작성]의 {공통 요청사항}을 준용하여 개발  - 'TDD이해'의 '통합 테스트 전략'을 준용하여 개발  - 요청된 {서비스}의 소스를 분석하여 통합 테스트가 필요한 클래스를 식별  - 각 layer별로 모든 통합 테스트 코드를 작성: controller, service, repository  - 다른 layer는 Mocking하여야 함  - Spring Security 사용 시 TestSecurityConfig 클래스에서 Security 설정 구성. @Profile로 적용된 프로파일 지정.  - 테스트 Class에 @ActiveProfile("integration-test")로 지정  - Controller 테스트 작성 시 유의사항:  1. 클래스 구조  - @WebMvcTest 사용 (@SpringBootTest 사용하지 않음)  - static TestConfig 클래스에서 Controller와 필요한 빈들을 직접 등록  - @Import({TestConfig.class, TestSecurityConfig.class}) 순서로 설정  2. TestConfig 작성  ```  @Configuration  static class TestConfig implements WebMvcConfigurer {  @Bean  public TestedController testedController() {  return new TestedController(requiredService());  }  @Bean  public RequiredService requiredService() {  return mock(RequiredService.class);  }  }  ```  3. TestSecurityConfig 작성: Spring Security 사용시  ```  @Configuration  @Profile("integration-test")  public class TestSecurityConfig {  @Bean  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  http  .csrf(AbstractHttpConfigurer::disable)  .authorizeHttpRequests(auth -> auth  .requestMatchers("public-urls").permitAll()  .anyRequest().authenticated()  );  return http.build();  }  }  ```  - Service 테스트 작성 시 유의사항:  1. 클래스 구조  - @ExtendWith(MockitoExtension.class) 사용 (@SpringBootTest 사용하지 않음)  - @Mock으로 외부 의존성 처리  - @InjectMocks로 테스트 대상 서비스 주입  - Spring Security 사용 시 @Import(TestSecurityConfig.class) 설정  2. 테스트 메소드 작성  ```  @Test  @DisplayName("테스트 시나리오 설명")  void givenCondition\_whenAction\_thenResult() {  // Given  given(mockDependency.method()).willReturn(value);    // When  Result result = service.method();    // Then  assertThat(result).isNotNull();  verify(mockDependency).method();  }  ```  - Repository 테스트 작성 시 유의사항:  1. 클래스 구조  - @DataJpaTest 사용 (@SpringBootTest 사용하지 않음)  - @TestInstance(TestInstance.Lifecycle.PER\_CLASS) 사용  - @AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE) 설정  - @Testcontainers 설정  - 실제 DB 사용을 위한 TestContainer 구성  2. TestContainer 설정: 'TestContainer 설정 가이드' 참조  <참고자료>  - TDD이해  - MergedSource  - TestContainer 설정 가이드  <응답순서>  - 테스트 코드 패키지와 클래스 구조도  - 테스트 클래스 선정 이유  - 테스트 코드: 각 클래스마다 별도의 코드블록으로 제공  <응답형식>  코드블록  {E2E 테스트 코드 작성}  'te:'로 시작하면 E2E 테스트 코드 작성 요청임  <요청사항>  - [테스트 코드 작성]의 {공통 요청사항}을 준용하여 개발  - 'TDD이해'의 'E2E 테스트 전략'을 준용하여 개발  - 요청된 {서비스}의 소스를 분석하여 통합 테스트 시나리오를 작성  - 테스트 Class에 @ActiveProfile("e2e-test")로 지정  - 유의사항:  1. 클래스 구조  - @SpringBootTest와 WebTestClient 사용  - 실제 DB 사용을 위한 TestContainer 구성을 TestContainerConfig에 구현하여 상속  - TestContainerConfig구현은 'TestContainer 설정 가이드' 참조  ```  @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)  @ActiveProfiles("e2e-test")  class ServiceE2ETest extends TestContainerConfig {  @Autowired  private WebApplicationContext context;    private WebTestClient webClient;  @BeforeEach  void setUp() {  webClient = MockMvcWebTestClient  .bindToApplicationContext(context)  .apply(SecurityMockMvcConfigurers.springSecurity())  .configureClient()  .build();  }  }  ```  2. TestDataManager 구성  ```  @Component  @RequiredArgsConstructor  public class TestDataManager {  private final Repository repository;  private final PasswordEncoder passwordEncoder;    @Value("${test.user.id}")  private String TEST\_USER\_ID;    @Transactional  public void setupTestData() {  // 테스트 데이터 생성 로직  }  }  ```  3. WebTestClient를 사용한 API 테스트  ```  @Test  @DisplayName("테스트 시나리오 설명")  void scenario() {  // Given  RequestDTO request = createRequest();  // When & Then  webClient.post().uri("/api/endpoint")  .contentType(MediaType.APPLICATION\_JSON)  .bodyValue(request)  .exchange()  .expectStatus().isOk()  .expectBody(ApiResponse.class)  .value(response -> {  assertThat(response.getStatus()).isEqualTo(200);  // ... 추가 검증  });  }  ```  <참고자료>  - TDD이해  - MergedSource  - TestContainer 설정 가이드  <응답순서>  - 테스트 코드 패키지와 클래스 구조도  - 테스트 시나리오와 목적  - 테스트 application-e2e-test.yml  - 테스트 코드: 각 클래스마다 별도의 코드블록으로 제공  <응답형식>  코드블록 |
| --- |

## 2) 프롬프트

- 단위 테스트

tu: {service}에 대한 단위 테스트 코드를 작성해 주세요.

o: instruction의 단위테스트 가이드라인을 꼼꼼히 보고 잘못된 부분이나 누락된 부분이 있는지 검토해 주세요.

- 통합 테스트

ti: {service}에 대한 통합 테스트 코드를 작성해 주세요.

o: instruction의 통합테스트 가이드라인을 꼼꼼히 보고 잘못된 부분이나 누락된 부분이 있는지 검토해 주세요.

- E2E 테스트

te: {member}서비스의 E2E 테스트 코드를 작성해 주세요.

o: instruction의 E2E 테스트 가이드라인을 꼼꼼히 보고 잘못된 부분이나 누락된 부분이 있는지 검토해 주세요.