데모 기획 및 개발 가이드

# 1. 데모기획 가이드

## 데모 기획하기

### 1. Instruction 추가

| [데모기획]  'dp:'로 시작하면 솔루션가설 검증을 위한 데모 기획 요청입니다.  {요청사항}  - '[최적화 가이드]'에 따라 제시된 {검증목적}을 위한 데모 시나리오를 작성함  - 사용패턴은 ‘Cloud Design Patterns(개요).docx’을 참조하여 분석  {참고자료}  - MergedSource  - MergedFront  - sample\_데모기획  - Cloud Design Patterns(개요).docx  {응답구성}  1. 목적: {검증목적}  2. 사용 패턴: application.yml에 정의된 Cloud Design Pattern  3. 사용 Backing서비스: application.yml과 build.gradle을 파악하여 실제 사용된 것만 제시  - Database  - MQ  - Blob Storage  4. 비즈니스 상황  5. 검증 지표와 카드:  - 섹션별로 카드를 제공  - 섹션과 카드명은 비즈니스 용어를 사용  - 각 카드는 그래프, 수치, 목표치를 제시 |
| --- |

### 2. 참고자료 업로드

- MergedSource: python3 mergedsrc.py

- sample\_데모기획

| 1. 목적  - 대규모 트래픽 상황에서의 실시간 사용량 조회 성능 검증  - 특히 응답시간, 처리량, 데이터 일관성을 중점적으로 검증  2. 사용 패턴  - Cache-Aside 패턴 (RedisConfig에 구현)  - Queue-Based Load Leveling 패턴 (EventHub 활용)  3. 사용 Backing서비스  - Database: PostgreSQL (application.yml의 datasource 설정)  - Cache: Redis (RedisConfig 설정)  - MQ: Azure Event Hub (EventHubConfig 설정)  4. 비즈니스 상황  - 명절 연휴 시작 시점의 데이터 사용량 조회 시나리오  - 전국 약 100만 가구가 동시에 사용량 조회 요청  - 가구당 평균 3개 디바이스에서 동시 접속  - 디바이스당 10초 간격으로 자동 갱신  - 초당 약 300,000건의 조회 요청 발생 예상  5. 검증 지표와 카드  Section1. 사용자 경험  1) 응답시간 분포  - 그래프: 응답시간 분포도(percentile)  - 수치: p95, p99 응답시간  - 목표치: p95 < 500ms, p99 < 1s  2) 처리량  - 그래프: 초당 처리 건수 추이  - 수치: 실시간/평균 TPS  - 목표치: 300,000 TPS  Section2. 시스템 성능  1) 캐시 효율성  - 그래프: 캐시 히트율 추이  - 수치: 히트율/미스율  - 목표치: 히트율 95% 이상  2) 리소스 사용률  - 그래프: CPU, 메모리 사용률 추이  - 수치: 평균/최대 사용률  - 목표치: CPU 70% 이하, 메모리 80% 이하 |
| --- |

- Cloud Design Patterns(개요).docx

<https://docs.google.com/document/d/1keYunbkDtMNjOWNDOEu6J0NWmEjixPIX/edit?usp=drive_link&ouid=110516422607493163049&rtpof=true&sd=true>

## 3. 프롬프트

아래 예와 같이 검증목적과 추가 필요한 사항을 제시함

| dp: 데모 기획을 해 주세요.  - {검증목적}: 대규모 트래픽 상황에서의 실시간 사용량 갱신 보장성 검증  - 초당 요청 건수: 350건 |
| --- |

결과를 보고 아래 사항을 클로딩하면서 완성해 나감

1) 클라우드 디자인 패턴 검토

* 각 패턴이 어떻게 구현된건지 소스를 보여주면서 설명해줘요.
* {의심스러운 패턴}은 적용했다고 하기엔 미약한거 같아요. 다시 검토 바랍니다.

예) Cache-Aside, Circuit Breaker, Throttling 패턴은 적용했다고 하기엔 미약한거 같아요. 다시 검토 바랍니다.

* {제외할 패턴}은 클라우드 디자인 패턴에 없는것이니 제외했으면 좋겠어요.  
  예) Sidecar/Microservices, Fallback은 클라우드 디자인 패턴에 없는것이니 제외했으면 좋겠어요. 그래서 사용하는 패턴만 반영해서 데모기획을 다시 해 주세요.

2) 메트릭 수집 방법 검토

* {메트릭 수집 방법이 궁금한 섹션 또는 카드}  
  예1) 'Queue-Based Load Leveling'의 메트릭을 수집할 방법이 있나요? 프론트엔드에서 수행되는 것이라 수집이 어렵지 않을까요?  
  예2) 'Gateway Routing'의 메트릭을 수집하는 방법은요? nginx ingress controller를 사용하고 있는데 어떻게 메트릭을 수집하죠?

3) 제외할 지표 요청

* {제외할 검증 지표}는 {제외 이유}로 제외해 주세요.   
  예) MVP목적에 비추어 'Retry'검증은 중요성이 떨어지니 제외해 주세요.

4) 선택과 집중: 목적에 비추어 가장 중요한 검증지표와 대시보드 카드 선택

* o: 검증지표와 카드를 목적에 비추어 가장 중요하다고 생각하는 섹션 2개와 각 섹션별 카드 2개만 선정해 주세요.

5) 최종 데모기획 완료:

* 클로딩한 결과와 팀의 생각을 종합하여 최종안을 작성
* MVP목적에 맞추어 가장 중요한 1~2가지 증명할 것을 선정하여 작성

**최종 데모기획 예시**

아래 예시는 검증지표는 3개이지만 검증하려는 것은 마지막 ‘서비스 응답성’임.

파일 업로드와 다운로드 건수는 요청 건수를 표시하려는 보조적 지표임.

| **통신사 CMS 데모 기획서**  1. 목적  대량의 파일 업로드와 다운로드 상황에서의 안정적 시스템 사용 검증  2. 사용 패턴   * **Queue-Based Load Leveling**: 업로드 요청을 큐에 넣어 처리함으로써 서버 부하 분산 (uploadMiddleware.js에서 구현) * **Gateway Routing**: API 요청을 적절한 백엔드 서비스로 라우팅 (SecurityConfig 및 프론트엔드 라우팅) * **Asynchronous Request-Reply**: 프런트엔드(클라이언트)와 백엔드 간 비동기로 요청과 응답을 분리하여 응답 시간을 단축 * **Valet Key**: 클라이언트가 특정 리소스에 제한된 직접 접근을 할 수 있도록 토큰을 사용하는 패턴 (SAS 토큰)   3. 사용 Backing서비스   * **Database**: PostgreSQL (authentication, content-management, token-management 모듈의 application.yml) * **Blob Storage**: Azure Blob Storage (content-management, token-management 모듈의 application.yml, BlobStorageClient) * **Spring Security**: 인증 및 권한 부여 관리 (SecurityConfig.java) * **Spring Data JPA**: 데이터 접근 계층 관리 (Repository 클래스들)   4. 비즈니스 상황   * 통신사의 마케팅 부서와 고객 지원 부서가 분기별로 대규모 콘텐츠(프로모션 자료, 사용자 가이드 등) 업데이트 진행 * 약 500명의 내부 직원이 동시에 시스템에 접속하여 콘텐츠 관리 * 각 직원은 평균 5-10개의 파일(1MB~10MB 사이즈)을 업로드 * 업로드된 콘텐츠는 전국 지점(약 1,000개 지점)에서 동시에 다운로드하여 활용 * 초당 약 100건의 업로드/다운로드 요청 발생 예상 * 특히 신규 요금제 출시나 프로모션 시작 시점(매월 1일)에 트래픽 집중 발생   5. 검증 지표와 카드  *검증 목적: 대용량 콘텐츠 파일의 업로드와 다운로드 발생 시 시스템 성능 검증*   1. **콘텐츠 업로드 건수**    * 그래프: 초당 업로드 처리 건수 추이    * 수치: 평균/최대 TPS    * 목표치: 100 TPS 안정적 처리 2. **콘텐츠 다운로드 건수**    * 그래프: 초당 다운로드 처리 건수 추이    * 수치: 평균/최대 TPS    * 목표치: 200 TPS 안정적 처리 3. 서비스 응답성    * 그래프: 동시 접속자 수 증가에 따른 대시보드 응답시간 변화    * 수치: 평균 응답시간, 최대 동시 접속자 수    * 목표치: 500명 동시 접속 시 평균 응답시간 < 2초 |
| --- |

**데모기획 수행 예시**

<https://claude.ai/share/dd29cce4-e96b-4a6f-be4c-3a3cf60d8ddf>

**결과를 ‘데모기획’이라는 이름으로 Knowledge에 등록합니다.**

# 2. Prometheus와 Grafana 설치

## NodePool 설치

1) Azure Portal에서 사용중인 AKS에 Node Pool을 추가합니다.  
설치를 위한 NodePool 추가   
- Node Pool명: monitor  
- Mode: 사용자   
- Node size: DS2\_v2   
- node count: 1~3   
  
agentpool에 생성된 Node의 label을 확인합니다.

| k get nodes --show-labels | grep monitor  aks-monitor-... Ready <none> 8s v1.30.9 agentpool=monitor,... ``` |
| --- |

다른 user service용 Pod가 배포(Scheduling)되지 않도록 taint설정을 합니다.

먼저 본인 팀의 AKS NAME을 셋팅합니다.

| export AKS\_NAME=unicorn-aks |
| --- |

| az aks nodepool update --cluster-name ${AKS\_NAME} --name monitor --node-taints dedicated=monitor:NoSchedule |
| --- |

Node의 Taint설정을 확인합니다.

| k describe node -l agentpool=monitor | grep -i taint Taints: dedicated=monitor:NoSchedule |
| --- |

## Instruction에 추가

| [Telemetry 설치]  'ti:'로 시작하면 Prometheus와 Grafana 설치 요청입니다.  <Prometheus 설치>  {요청사항}  - Helm chart로 Prometheus를 설치하는 방법 안내  - 네임스페이스 'monitor'에 설치  - {Label}의 값이 {Values}중 하나인 Pod의 메트릭만 수집되도록 함  - alertmanager, kubeStateMetrics, nodeExporter는 비활성화  - server.service: type은 'LoadBalancer', servicePort는 '80'으로 함  - server.persistentVolume: 10Gi  - 수집주기(server.global.scrape\_interval): 10s  - 규칙평가 주기(server.global.evaluation\_interval): 10s  - ServiceMonitor 사용하지 않고 수집되게 함  - Helm chart는 'prometheus-community/prometheus' 사용. 최신 버전 사용  - values.yaml에 아래 내용 추가.  ```  nodeSelector:  agentpool: monitor  tolerations:  - key: "dedicated"  operator: "Equal"  value: "monitor"  effect: "NoSchedule"  ```  {응답순서}  - 작업 디렉토리 생성 및 이동: '~/install/prometheus'  - 네임스페이스 생성 및 이동: 이동 시 kubens 사용  - values.yaml 내용  - 설치 명령: 'helm upgrade' 사용  <Grafana 설치>  {요청사항}  - Helm chart로 Grafana를 설치하는 방법 안내  - 네임스페이스 'monitor'에 설치  - Helm chart는 'grafana/grafana' 사용. 최신 버전 사용.  - values.yaml에 아래 내용 추가.  ```  nodeSelector:  agentpool: monitor  tolerations:  - key: "dedicated"  operator: "Equal"  value: "monitor"  effect: "NoSchedule"  ```  {응답순서}  - 작업 디렉토리 생성 및 이동: '~/install/grafana'  - 네임스페이스 생성 및 이동: 이동 시 kubens 사용  - values.yaml 내용  - 설치 명령: 'helm upgrade' 사용  - 로그인 ID/PW  - datasource 생성 안내  - name: Prometheus  - url: <http://prometheus-server> |
| --- |

## 가이드 요청 프롬프트

1) 설치치

수집할 대상 Pod만을 지정합니다.

서비스명은 Deployment 또는 Statefulset의 app 레이블 값을 지정해야 합니다.

Values에 프론트엔드와 백엔드의 label값을 모두 넣으십시오.

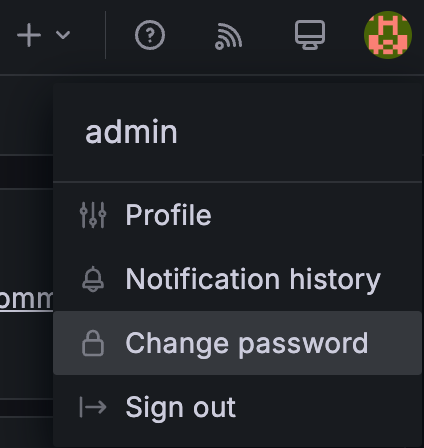
| ti: Prometheus와 Grafana 설치를 안내해 주세요.  - {Label}: app  - {Values}: usage-query-service, usage-update-service, usage-front |
| --- |

Values에 넣는 값은 아래 명령으로 label과 values를 확인해서 정확히 넣으세요.

| k get po --show-labels |
| --- |

2) Grafana 암호 변경

Grafana 로그인 후 암호를 변경합니다.



# 3. 메트릭 수집 처리

## Instruction에 추가

| [데모개발]  'dd:'로 시작하면 데모 개발 요청입니다.  <메트릭 수집 처리>  {요청사항}  - {서비스}에 '데모기획'의 '검증 지표와 카드'를 위한 메트릭 수집 처리 개발  - 멀티 프로젝트임을 고려: 공통 클래스는 common에 위치, 공통 Dependency는 root의 build.gradle에만 위치  - Prometheus로 Metric을 수집하여 Grafana로 가시화할 수 있도록 처리  - MetricsConfig 클래스는 common 프로젝트가 아니라 필요한 각 서비스에 있어야 함  - Metrics 이름에는 언더바(underbar)를 사용  - Timer를 생성할 때 SLA 구간(bucket)을 명시적으로 지정하여 histogram metrics 생성  {참고자료}  - MergedSource  - MergedFront  - 데모기획  - sample\_metrics  {응답순서}  - build.gradle: root project와 각 서비스별 build.gradle에 추가할 사항 제시  - application.yml: 각 서비스별 application.yml에 추가할 사항 제시.  - tags는 'app'와 'instance'로 함. 'application'이 아님에 주의.  - app: ${spring.application.name}  - instance: ${HOSTNAME}:${server.port}  - MetricsConfig 클래스: 한 곳에서 필요한 Metric을 관리하도록 필요한 class 생성  - Counter class 생성  - SLA 구간(bucket)을 명시적으로 지정한 Timer class 생성  - Service Layer 클래스: Metric 수집 처리  - 기타: 그 외 Metric 수집 처리를 위해 추가/수정해야 하는 클래스  - Workload controller manifest의 Annotation 추가  port는 application.yml의 server.port와 동일해야 함을 볼드체로 명시  ```  annotations:  prometheus.io/scrape: "true"  prometheus.io/path: "/actuator/prometheus"  prometheus.io/port: "8080"  ```  - Metrics 목록과 설명: 각 Metric이름별로 Prometheus에서 변환되는 Metrics명과 설명 제공  ```  # cache.operation.time  cache\_operation\_time\_seconds\_count : 전체 요청 수  cache\_operation\_time\_seconds\_sum : 전체 소요 시간의 합  cache\_operation\_time\_seconds\_max : 최대 소요 시간  cache\_operation\_time\_seconds\_bucket{le="0.1"} : 100ms 이하 요청 수  cache\_operation\_time\_seconds\_bucket{le="0.5"} : 500ms 이하 요청 수  cache\_operation\_time\_seconds\_bucket{le="1"} : 1s 이하 요청 수  ```  {응답형식}  코드블록  <대시보드 개발>  {요청사항}  - '데모기획'의 '검증 지표와 카드'를 위한 Grafana 대시보드를 개발  - '매트릭 목록'에 명시된 metric을 사용  - datasource.uid는 'Prometheus'로 함  - 제공된 {대시보드명}으로 생성  - 각 카드의 타이틀은 한글로 함  {참고자료}  - 데모기획  - Metrics 목록  {응답순서}  - 대시보드 json  - 대시보드 생성 방법  - Data source 등록 방법  <부하 스크립트 개발>  {요청사항}  - {서비스}의 {API}를 사용하여 k6 스크립트를 작성함  - 파일명: {서비스}명과 동일  - {Key}필드 처리: 하위에 제공된{format}을 지켜 {start}~{max}사이의 값을 순차적으로 생성  - {Random} 필드 처리: 하위에 제공된 {start}~{max}사이의 값을 랜덤하게 생성  - {Option} 필드 처리: 하위에 제공된 {values}의 값 중 하나를 랜덤하게 생성  {참고자료}  - MergedSource  {응답순서}  - k6 스크립트  - 테스트 방법: 'k6 run --iterations 1 {서비스}'  - 실행 방법  <테스트 페이지 개발>  {요청사항}  - API 호출 횟수를 지정하는 필드 제공. 기본값은 100번으로 함.  - '테스트 시작', '테스트 중지' 버튼 제공  - {Key}필드 처리: 하위에 제공된{format}을 지켜 {start}~{max}사이의 값을 순차적으로 생성  - {Random} 필드 처리: 하위에 제공된 {start}~{max}사이의 값을 랜덤하게 생성  - {Option} 필드 처리: 하위에 제공된 {values}의 값 중 하나를 랜덤하게 생성  - Client에서 0.5초 마다 API호출 횟수까지 계속 API를 호출하도록 함  - 응답속도를 텍스트로 크게 강조하여 표시  - 요청 데이터와 응답 데이터 표시  - React/Vue가 아닌 기본 HTML로 개발  - Material UI를 사용  - 반응형 디자인으로 개발하며 모바일에 최적화 함  {참고자료}  - MergedSource  - MergedFront  {응답구성}  - 프론트 페이지 디렉토리 작성 안내: ~/workspace/stress/{페이지ID}  - 페이지 소스:  - index.html  - app.js  - styles.css  - 수행 가이드  ```  cd ~/workspace/stress/{페이지ID}  python3 -m http.server {포트}  ```  브라우저에서 http://localhost:{포트}/index.html로 접근 안내  {응답형식}  코드블록 |
| --- |

## 참고자료 업로드

- MergedSource: ‘mergedsrc.py’로 작성

- MergedFront: ‘merge-node.py’로 작성

- 데모 기획: ‘1. 데모기획 가이드’의 결과 업로드

- sample\_metrics

| package com.ktds.krater.query.config;  import io.micrometer.core.instrument.MeterRegistry;  import io.micrometer.core.instrument.Timer;  import io.micrometer.core.instrument.simple.SimpleMeterRegistry;  import io.micrometer.core.instrument.Counter;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import java.time.Duration;  @Configuration  public class MetricsConfig {  @Bean  public MeterRegistry meterRegistry() {  return new SimpleMeterRegistry();  }  @Bean  public Timer usageQueryTimer(MeterRegistry registry) {  return Timer.builder("usage\_query\_time")  .description("Time taken to query usage")  .publishPercentiles(0.5, 0.95, 0.99)  .publishPercentileHistogram()  .sla(Duration.ofMillis(500), Duration.ofMillis(1000))  .register(registry);  }  @Bean  public Counter usageRequestCounter(MeterRegistry registry) {  return Counter.builder("usage\_requests\_total")  .description("Total number of usage requests")  .register(registry);  }  @Bean  public Counter usageErrorCounter(MeterRegistry registry) {  return Counter.builder("usage\_errors\_total")  .description("Total number of usage request errors")  .register(registry);  }  @Bean  public Counter cacheHitCounter(MeterRegistry registry) {  return Counter.builder("usage\_cache\_hits\_total")  .description("Total number of cache hits")  .register(registry);  }  @Bean  public Counter cacheMissCounter(MeterRegistry registry) {  return Counter.builder("usage\_cache\_misses\_total")  .description("Total number of cache misses")  .register(registry);  }  }  package com.ktds.krater.query.service;  import com.ktds.krater.common.constants.CacheConstants;  import com.ktds.krater.common.dto.UsageDTO;  import com.ktds.krater.common.entity.Usage;  import com.ktds.krater.query.mapper.UsageMapper;  import com.ktds.krater.query.repository.UsageRepository;  import io.micrometer.core.instrument.Counter;  import io.micrometer.core.instrument.Timer;  import lombok.RequiredArgsConstructor;  import lombok.extern.slf4j.Slf4j;  import org.springframework.data.redis.core.RedisTemplate;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;  import java.time.Duration;  @Slf4j  @Service  @RequiredArgsConstructor  public class UsageQueryService {  private final UsageRepository usageRepository;  private final UsageMapper usageMapper;  private final RedisTemplate<String, UsageDTO> redisTemplate;  private final Timer usageQueryTimer;  private final Counter usageRequestCounter;  private final Counter usageErrorCounter;  private final Counter cacheHitCounter;  private final Counter cacheMissCounter;  @Transactional(readOnly = true)  public UsageDTO getUserUsage(String userId) {  Timer.Sample sample = Timer.start();  try {  String cacheKey = CacheConstants.USAGE\_CACHE\_PREFIX + userId;  // 캐시 시도  UsageDTO cachedUsage = redisTemplate.opsForValue().get(cacheKey);  if (cachedUsage != null) {  cacheHitCounter.increment();  return cachedUsage;  }  cacheMissCounter.increment();  // DB 조회  Usage usage = usageRepository.findByUserId(userId)  .orElseThrow(() -> new IllegalArgumentException("User not found: " + userId));  UsageDTO usageDTO = usageMapper.toDTO(usage);  // 캐시 업데이트  redisTemplate.opsForValue().set(cacheKey, usageDTO, Duration.ofMinutes(30));  sample.stop(usageQueryTimer);  usageRequestCounter.increment();  return usageDTO;  } catch (Exception e) {  usageErrorCounter.increment();  log.error("Error getting usage for userId: {}", userId, e);  throw e;  }  }  } |
| --- |

## 가이드 요청 프롬프트

메트릭 수집 처리를 추가/수정할 서비스 명을 입력하세요.

| dd: 메트릭 수집 처리를 개발해 주십시오. 수정이 필요한 부분을 자세하게 가이드 해 주세요.  - {서비스}: usage-query-service |
| --- |

만약 추가/수정할 서비스명을 잘 모를때는 ‘{서비스}’를 지정 안하셔도 됩니다.

AI 팀원들이 소스를 분석하여 수정 할 부분을 안내해 줄겁니다.

**개발 가이드 예제**

[**https://claude.ai/share/4141eec3-ac71-42ac-876a-297859545731**](https://claude.ai/share/4141eec3-ac71-42ac-876a-297859545731)

# 4. 대시보드 개발

Grafana 대시보드를 개발 합니다.

## Instruction 추가

이미 ‘3.메트릭 수집 처리’에서 추가했습니다.

## 참고자료 업로드

- 데모 기획: ‘1. 데모기획 가이드’의 결과 업로드

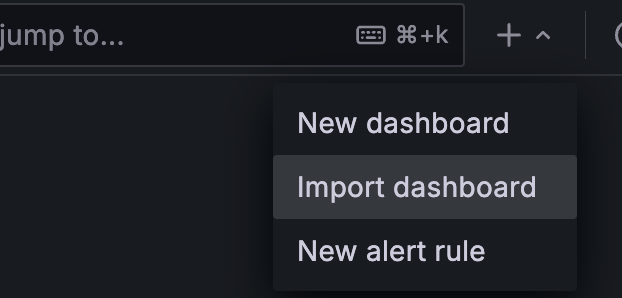
- Metrics 목록: ‘3. 메트릭 수집 처리’의 마지막 결과를 업로드

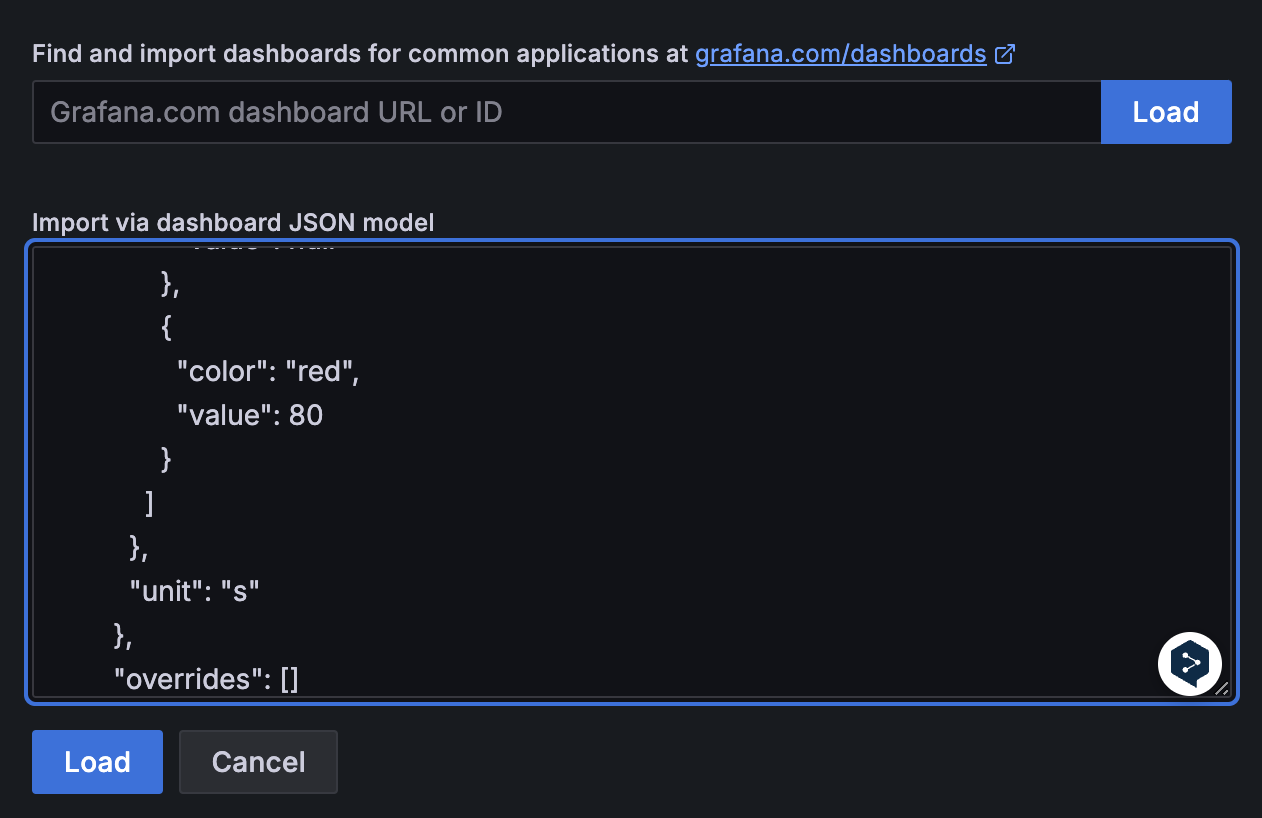
## 가이드 요청 프롬프트

대시보드명은 대시보드의 성격에 맞게 지정해 주세요.

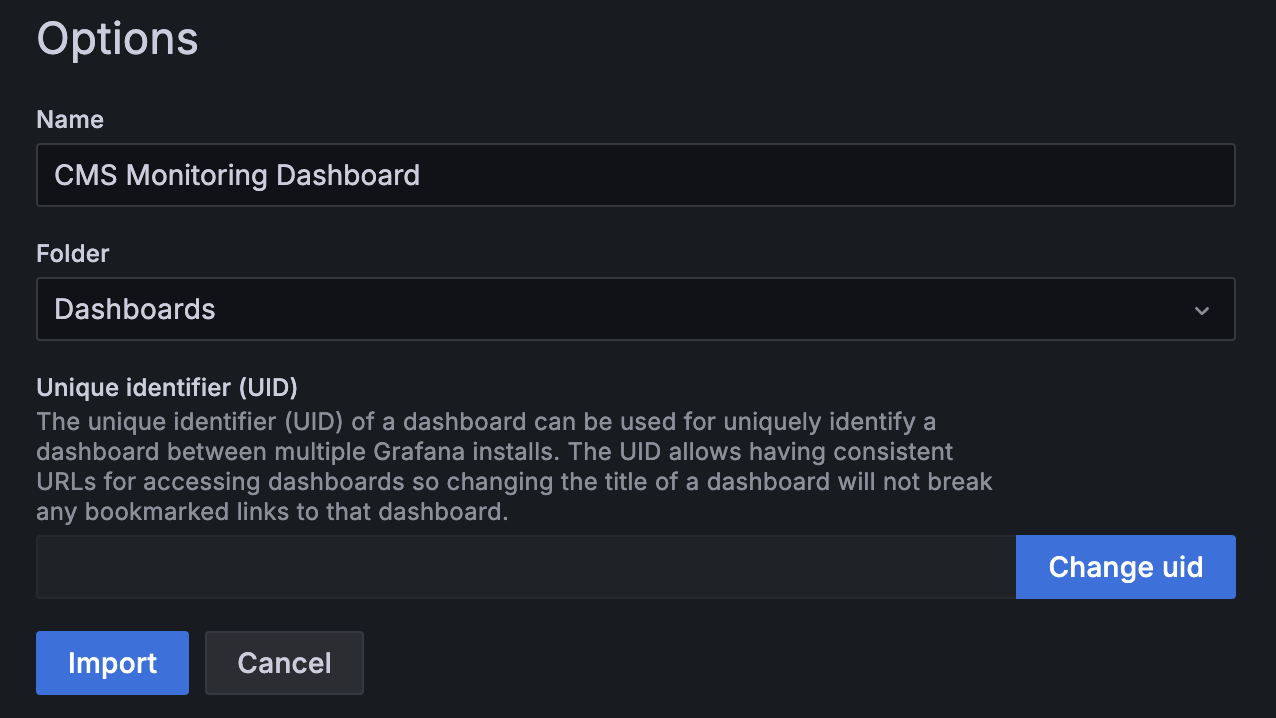
| dd: 대시보드 개발을 해주세요.  - {대시보드명}: 사용량 조회 |
| --- |

Json파일 내용을 아래 화면과 같이 붙여넣어 대시보드를 만듭니다.





Load 클릭 후 Import 클릭



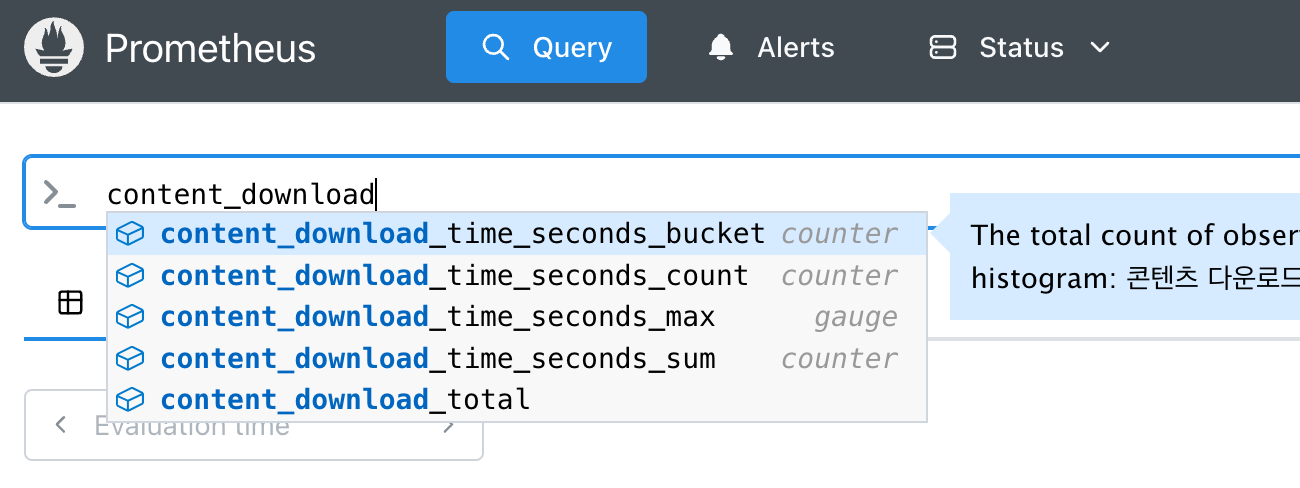
## Metric 수집 체크

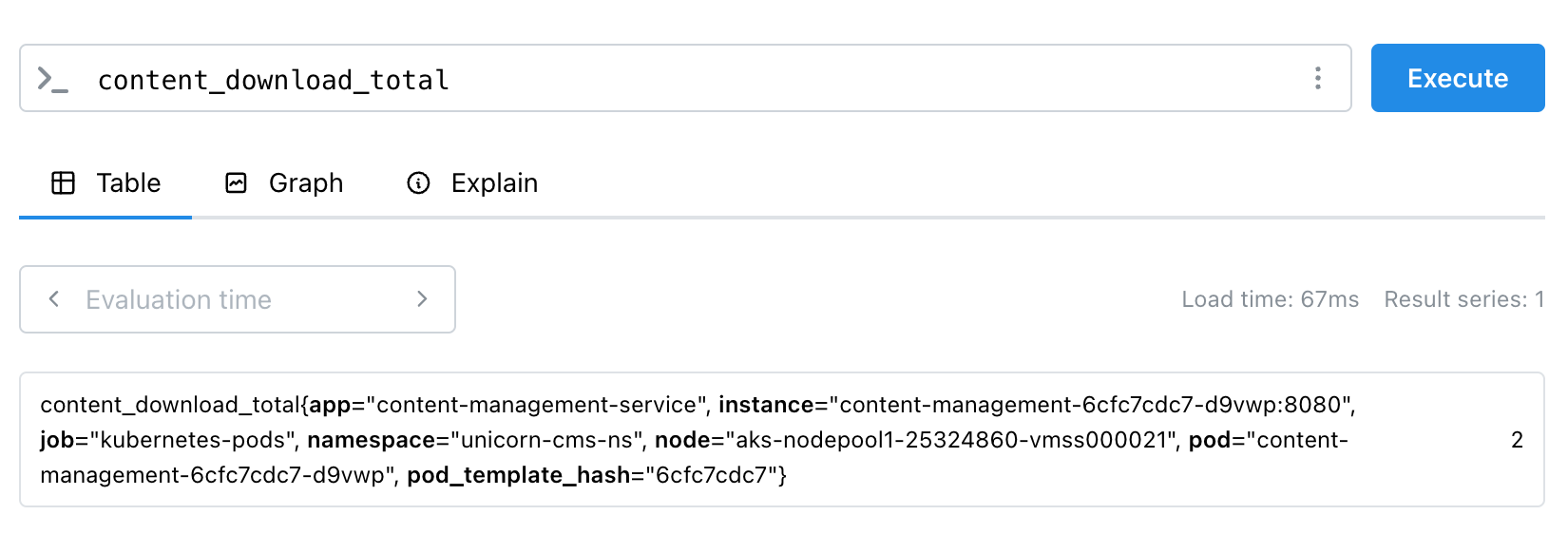
**1) Prometheus 수집 체크**

서비스 ‘prometheus-server’의 External IP로 브라우저에서 접근합니다.

MVP서비스에서 metric이 수집되는 Action을 수행합니다.

수집할 metric이름으로 검색하여 수집이 되는지 확인합니다.

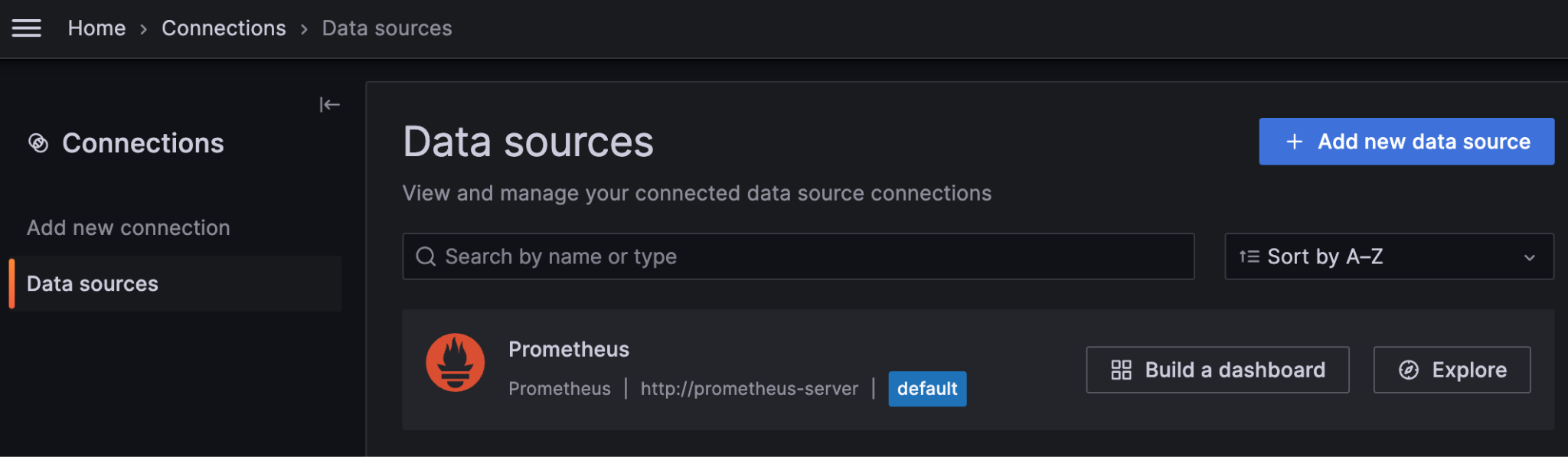




**2) Grafana metric 수집 체크**

Grafana에서 아래와 같이 체크합니다.

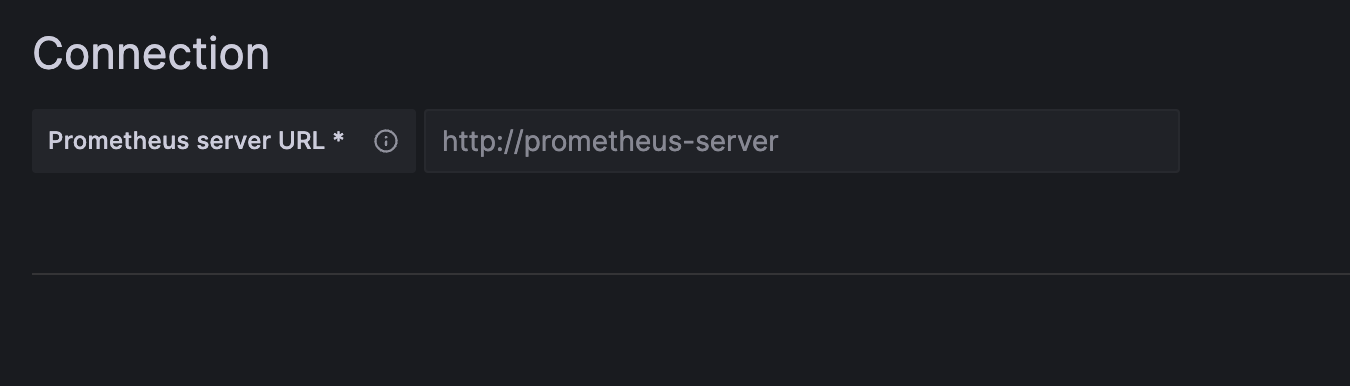
좌측 상단의 햄버거 메뉴에서 Connections > Data sources를 클릭합니다.



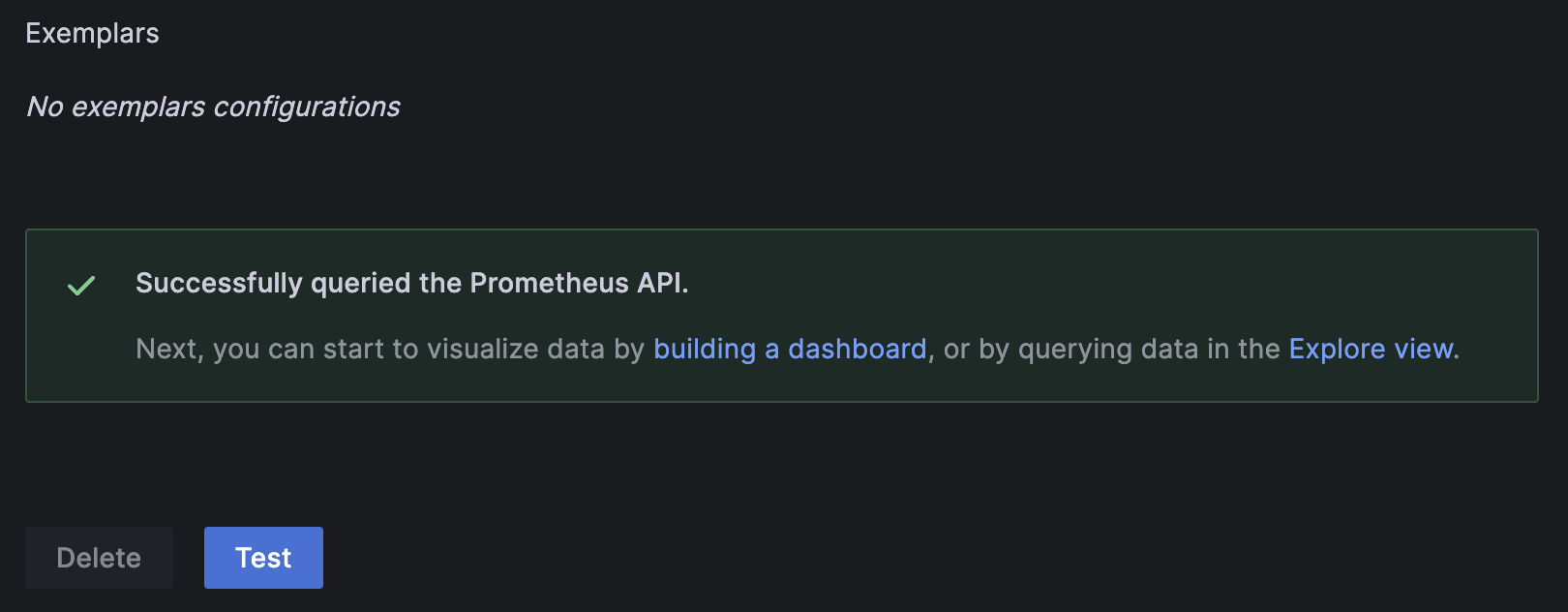
‘Prometheus’를 클릭하여 들어갑니다.

만약에 없으면 새로 만듭니다.

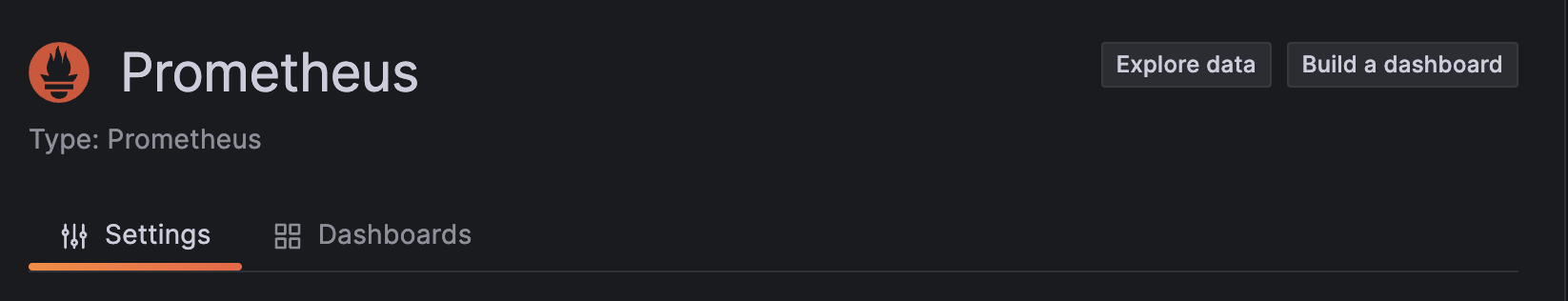
아래와 같이 서버 주소를 Prometheus의 서비스 오브젝트 주소로 되어 있는지 확인합니다.



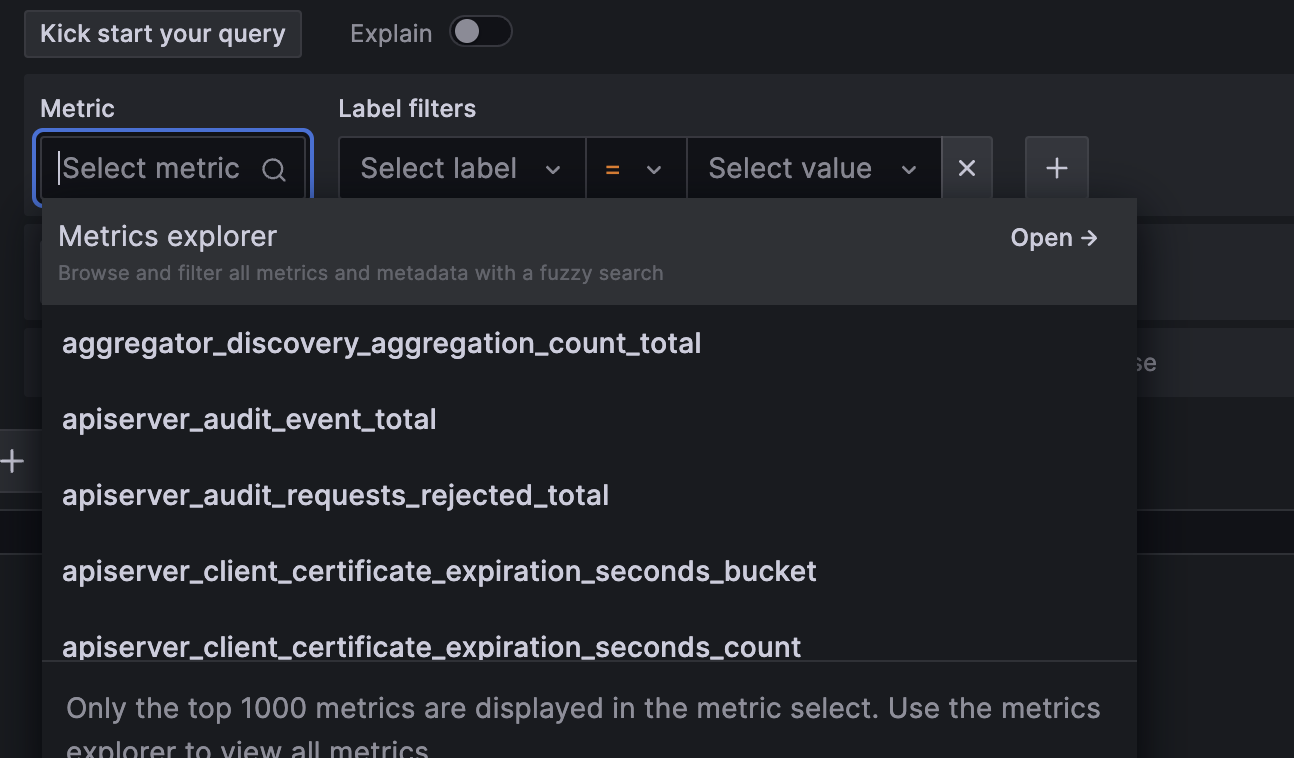
제일 하단의 ‘Test’버튼을 눌러 Prometheus와 연결이 되는지 확인합니다.



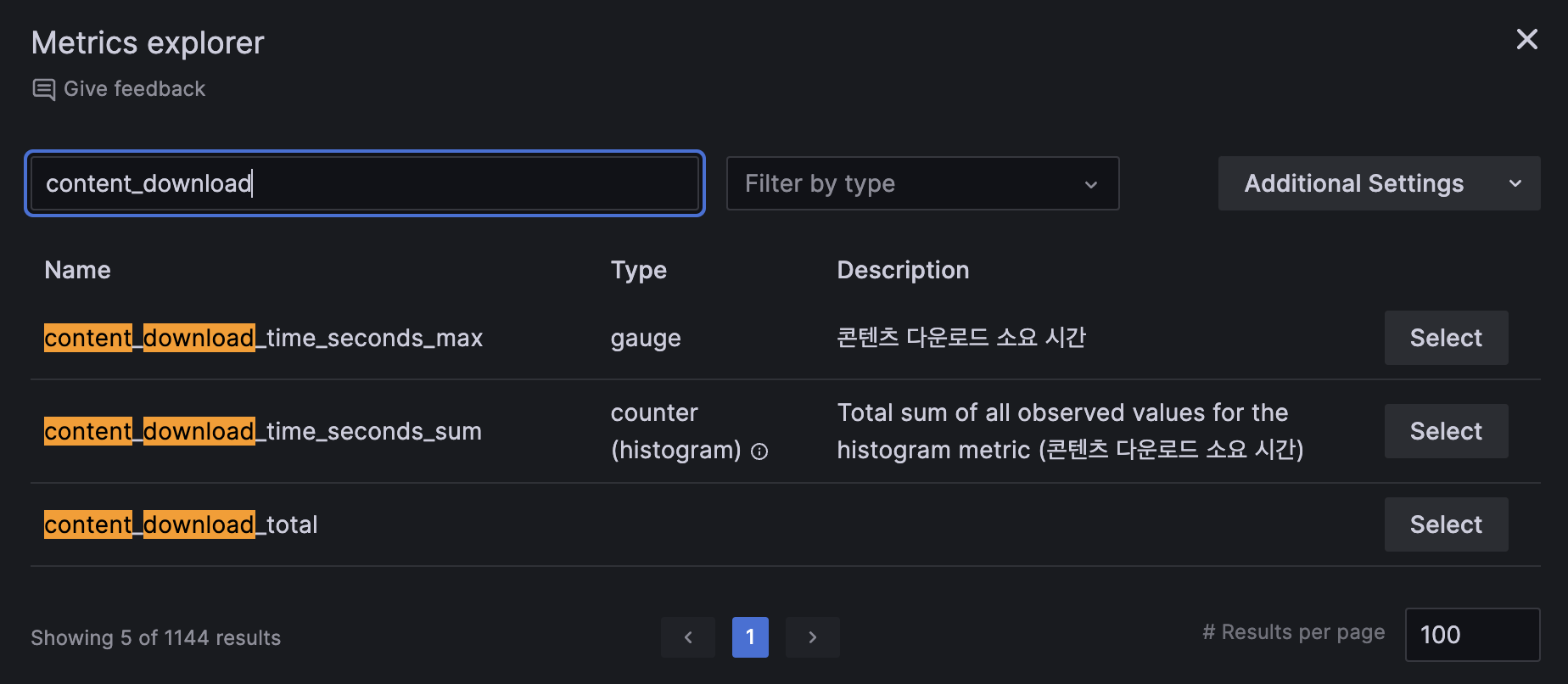
제일 상단의 ‘Explore data’ 버튼을 클릭합니다.



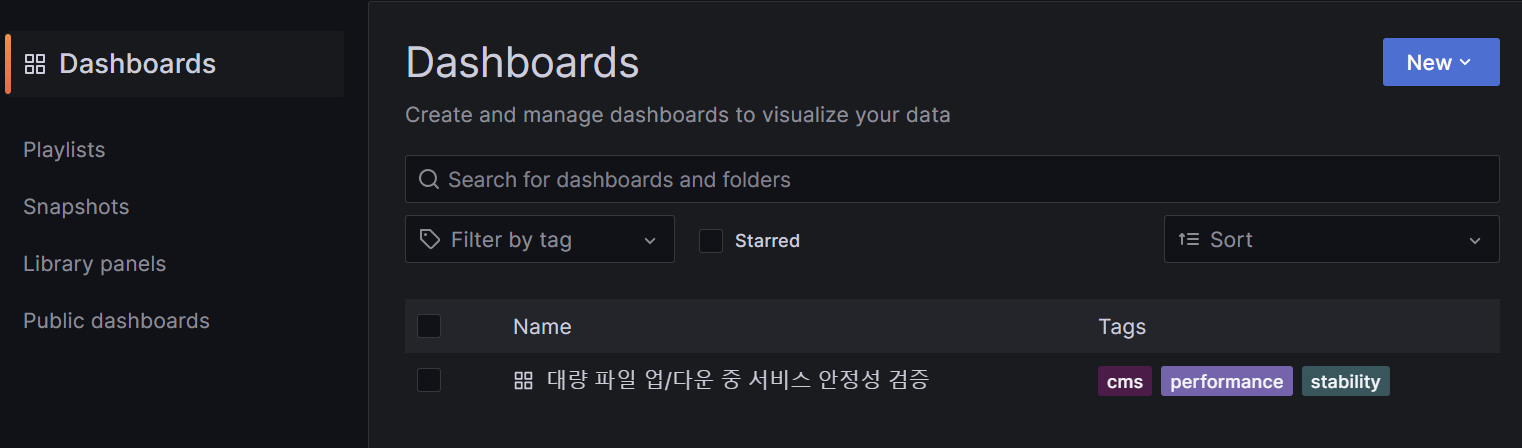
Metric 필드를 클릭하고 ‘Metrics explorer’을 선택합니다.

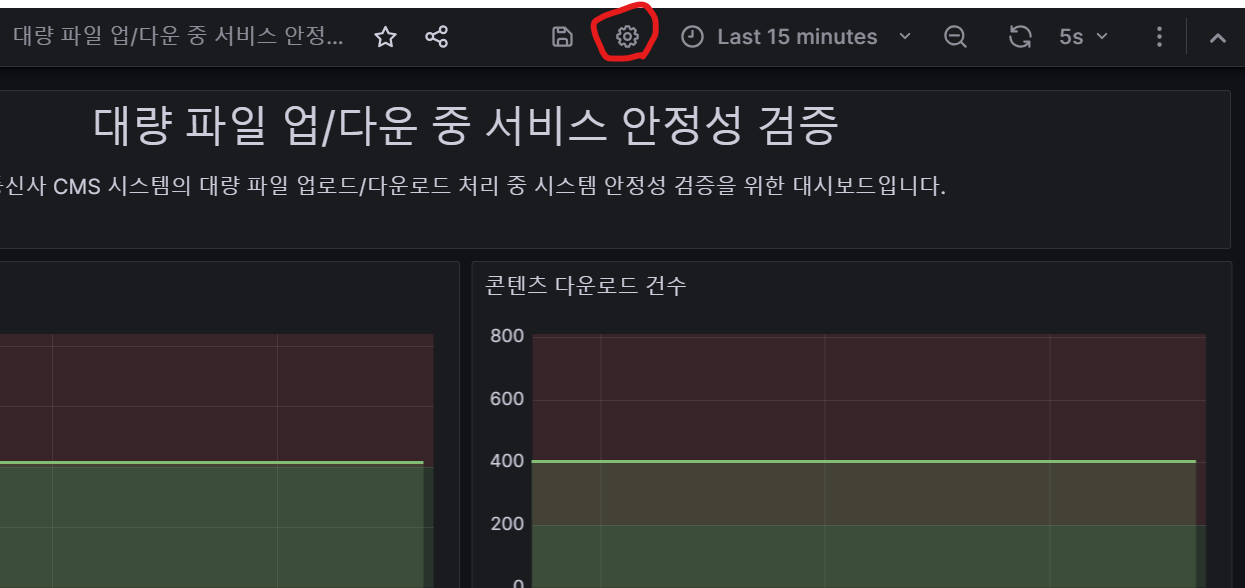


수집할 metric 이름을 검색하여 나오는지 확인 합니다.



## 대시보드 백업하기

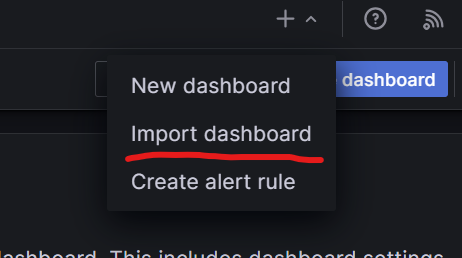






위 JSON내용을 백업합니다.

Restore는 대시보드 만들때처럼 상단 우측에서 ‘Import dashboard’한 후에 JSON을 붙여넣으면 됩니다.



# 5. 부하 스크립트 개발

K6로 부하 스크립트를 개발하는 방법을 안내 합니다.

복합 시나리오로 부하를 발생 하려면 jMeter 가이드를 참고하세요.

<https://docs.google.com/document/d/1haR0NR-19Nd6YWBJx6C64xkjkcdUx-d9mP_aHceeIBY/edit?usp=drive_link>

## Instruction 추가

이미 ‘3.메트릭 수집 처리’에서 추가했습니다.

## 참고자료 업로드

- MergedSource: ‘mergedsrc.py’로 작성

## 가이드 요청 프롬프트

아래 예와 같이 API에 따라 적절히 지정하세요.

- {Key}: 1씩 증가시킬 key 필드

- {Option}: 리스트에서 랜덤하게 1개 값을 선택

- {Random}: 일정 범위 내에서 랜덤하게 부여

- FQDN: 부하를 주기 위해 서비스를 접근할 수 있는 주소. [http://{ingress](about:blank) host}/{path}형태

결국 부하는 FQDN/{API경로}로 부하가 감

예1)

| dd: 부하 스크립트를 개발해 주세요.  - {서비스}: usage-query-service  - FQDN: http://4.230.151.165/api/member  - {API}: getUserUsage  - Method: GET  - {Key}: userId  - {format}: user0000  - {start}: 1  - {max}: 1000 |
| --- |

예2)

| dd: 부하 스크립트를 개발해 주세요.  - {서비스}: usage-update-service  - {API}: getUserUsage  - Method: POST  - FQDN: http://4.230.151.165/api/member  - {Key}: userId  - {format}: user0000  - {start}: 1  - {max}: 1000  - {Option}: type  - {values}: VOICE, DATA, MESSAGE, DATA  - {Random}: amount  - {start}: 100  - {max}: 10000 |
| --- |

예3)

| dd: 부하 스크립트를 개발해 주세요.  - {서비스}: authentication  - FQDN: http://20.249.205.95.nip.io/api/auth/login  - Method: POST  - {Key}: username  - {format}: user00  - {start}: 1  - {max}: 5  - 암호: P@ssw0rd$  - Post Data 예시: {username: "user01", password: "P@ssw0rd$"} |
| --- |

예4)

| dd: 부하 스크립트를 개발해 주세요.  - {서비스}: content-management  - FQDN: http://20.249.205.95.nip.io/api/contents  - Method: POST  - {Key}: username  - {format}: user00  - {start}: 1  - {max}: 5  -Request Header 'Authorization'에 아래값 추가  Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMDEiLCJyb2xlcyI6WyJVU0VSIl0sImlhdCI6MTc0MjQwMDkzMiwiZXhwIjoxNzQyNDA0NTMyfQ.-T9ZmhP1vkgMjhpWbmGCKuDy3iLZzKYE4phsV9lvhlI  -Post Data:  - 예시: {contentName: "상품소개서", contentType: "marketing-materials", description: "", fileName: "제안서.pdf"}  - {Key}: contentName  - {format}: Test0000  - {start}: 1  - {max}: 1000  - {Option}: contentType  - {values}: marketing-materials, user-guides  - {Key}: fileNameName  - {format}: File0000  - {start}: 1  - {max}: 1000 |
| --- |

예5)

| dd: 부하 스크립트를 개발해 주세요.  - {서비스}: content-management  - FQDN: http://20.249.205.95.nip.io/api/contents/{filenum}  - Method: GET  - {Random}: filenum  - {start}: 1  - {max}: 100  - Request Header 'Authorization'에 아래값 추가  Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMDEiLCJyb2xlcyI6WyJVU0VSIl0sImlhdCI6MTc0MjQwMDkzMiwiZXhwIjoxNzQyNDA0NTMyfQ.-T9ZmhP1vkgMjhpWbmGCKuDy3iLZzKYE4phsV9lvhlI |
| --- |

**프롬프트 결과 예시**

<https://claude.ai/share/101da694-1b1e-4600-a271-c2127f7c44b4>

## K6s 스크립트 실행

Local Ubuntu 또는 VM에 k6를 설치합니다.

| # GPG key 추가  sudo gpg -k  sudo gpg --no-default-keyring --keyring /usr/share/keyrings/k6-archive-keyring.gpg --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys C5AD17C747E3415A3642D57D77C6C491D6AC1D69  # k6 repository 추가  echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list  # 시스템 패키지 업데이트  sudo apt-get update  # k6 설치  sudo apt-get install k6  # 설치 확인  k6 version |
| --- |

테스트 후 실행 합니다.

- 테스트

| k6 run {script file} --iterations=1 |
| --- |

- 실행

| k6 run {script file} |
| --- |

# 튜닝 가이드

**프롬프트**

| o: MergedSource를 보고 Database와 MQ의 성능 튜닝 가이드를 주세요.  구체적으로 소스의 어디를 수정해야 하는지 알려 주세요. |
| --- |

## 1. Database 튜닝

Pod로 설치한 RDB 성능이 잘 안나오면 VM에 설치하세요.

개발가이드의 ‘5. Database설치(VM)’ 참고.

<https://docs.google.com/document/d/15q4cLdK7quBlm8nZhm6vNCKKtNPH1SfYb_yTLwaQSyQ/edit?tab=t.0#heading=h.iv358npaaioc>

- Index 만들기: Repository클래스의 key에 대해 생성

- Connection Pool 증가

- Batch로 Data update

## 2. MQ 튜닝

- Queue 갯수 늘리기

- Queue 할당 Key(예: partitionKey, Routing Key 등) 지정

# 6. 테스트 페이지 개발

부하 테스트를 요청할 프론트엔드를 간단하게 개발할때 사용하세요.

## Instruction 추가

이미 ‘3.메트릭 수집 처리’에서 추가했습니다.

## 참고자료 업로드

- MergedSource: ‘mergedsrc.py’로 작성

- MergedFront: ‘merge-node.py’로 작성

## 가이드 요청 프롬프트

POST 요청 예시

| dd: 프론트엔드 페이지를 개발해 주세요.  - 페이지 타이틀: 사용량 갱신  - 페이지 ID: update\_usage  - 포트: 3000  - FQDN: http://4.230.148.226/api/update  - Method: POST  - {Key}: userId  - {format}: user0000  - {start}: 1  - {max}: 1000  - {Option}: type  - {values}: VOICE, DATA, MESSAGE, DATA  - {Random}: amount  - {start}: 100  - {max}: 10000 |
| --- |

GET 요청 예시

| dd: 프론트엔드 페이지를 개발해 주세요.  - 페이지 타이틀: 사용량 조회  - 페이지 ID: query\_usage  - 포트: 3000  - FQDN: http://4.230.151.165/api/query  - Method: GET  - {Key}: userId  - {format}: user0000  - {start}: 1  - {max}: 1000 |
| --- |

예)

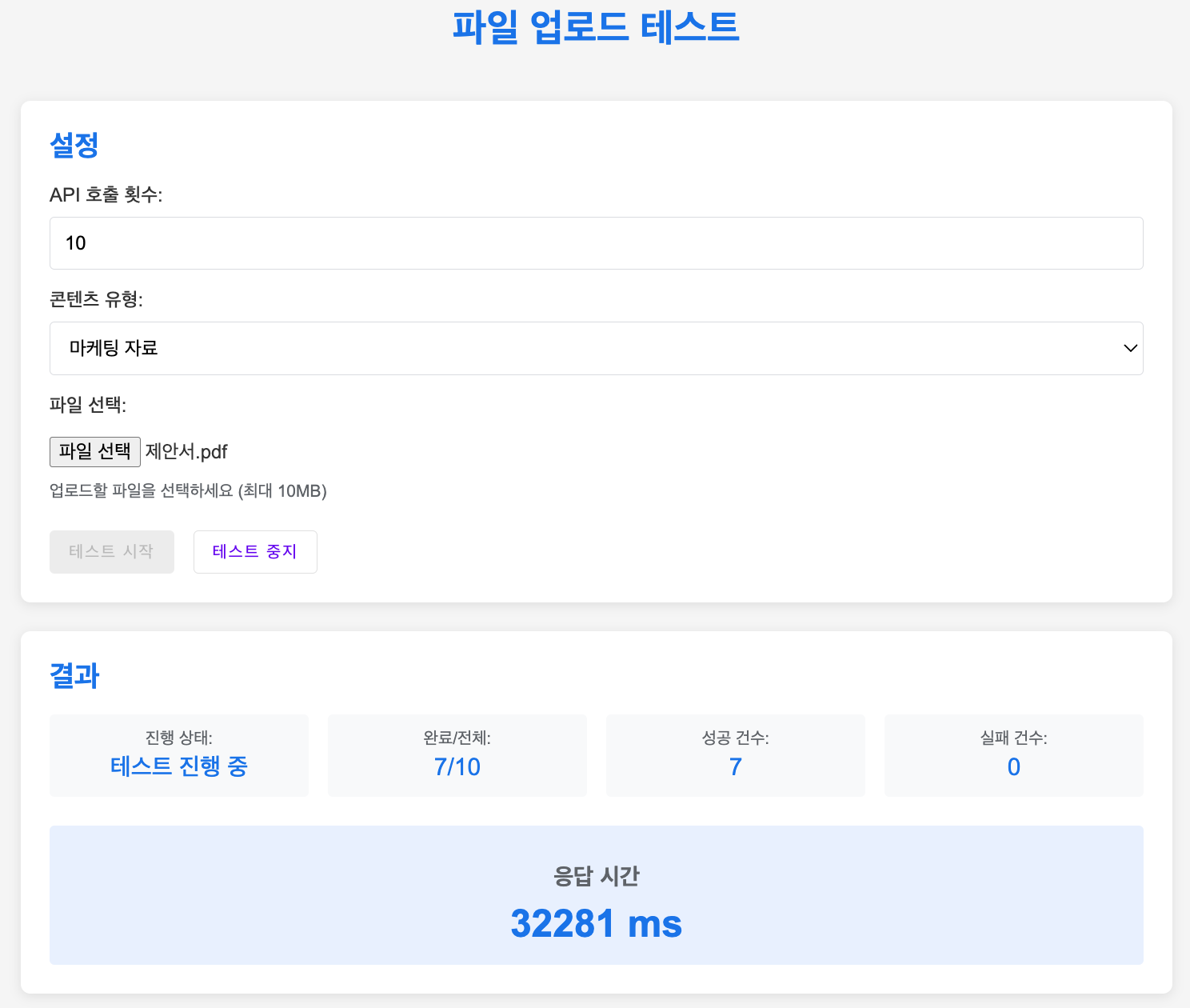
| dd: 테스트 페이지를 개발해 주세요.  실제 파일이 업로드 되도록 해주세요.  - 페이지 타이틀: 파일 업로드  - 페이지 ID: upload\_tester  - 포트: 3000  - FQDN: http://20.249.205.95.nip.io/api/contents  - Method: POST  - {Key}: username  - {format}: user00  - {start}: 1  - {max}: 5  - Request Header 'Authorization'에 아래값 추가  Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMDEiLCJyb2xlcyI6WyJVU0VSIl0sImlhdCI6MTc0MjQwMDkzMiwiZXhwIjoxNzQyNDA0NTMyfQ.-T9ZmhP1vkgMjhpWbmGCKuDy3iLZzKYE4phsV9lvhlI  - Post Data:  - 예시: {contentName: "상품소개서", contentType: "marketing-materials", description: "", fileName: "제안서.pdf"}  - {Key}: contentName  - {format}: Test0000  - {start}: 1  - {max}: 1000  - {Option}: contentType  - {values}: marketing-materials, user-guides  - {Key}: fileNameName  - {format}: File0000  - {start}: 1  - {max}: 1000 |
| --- |

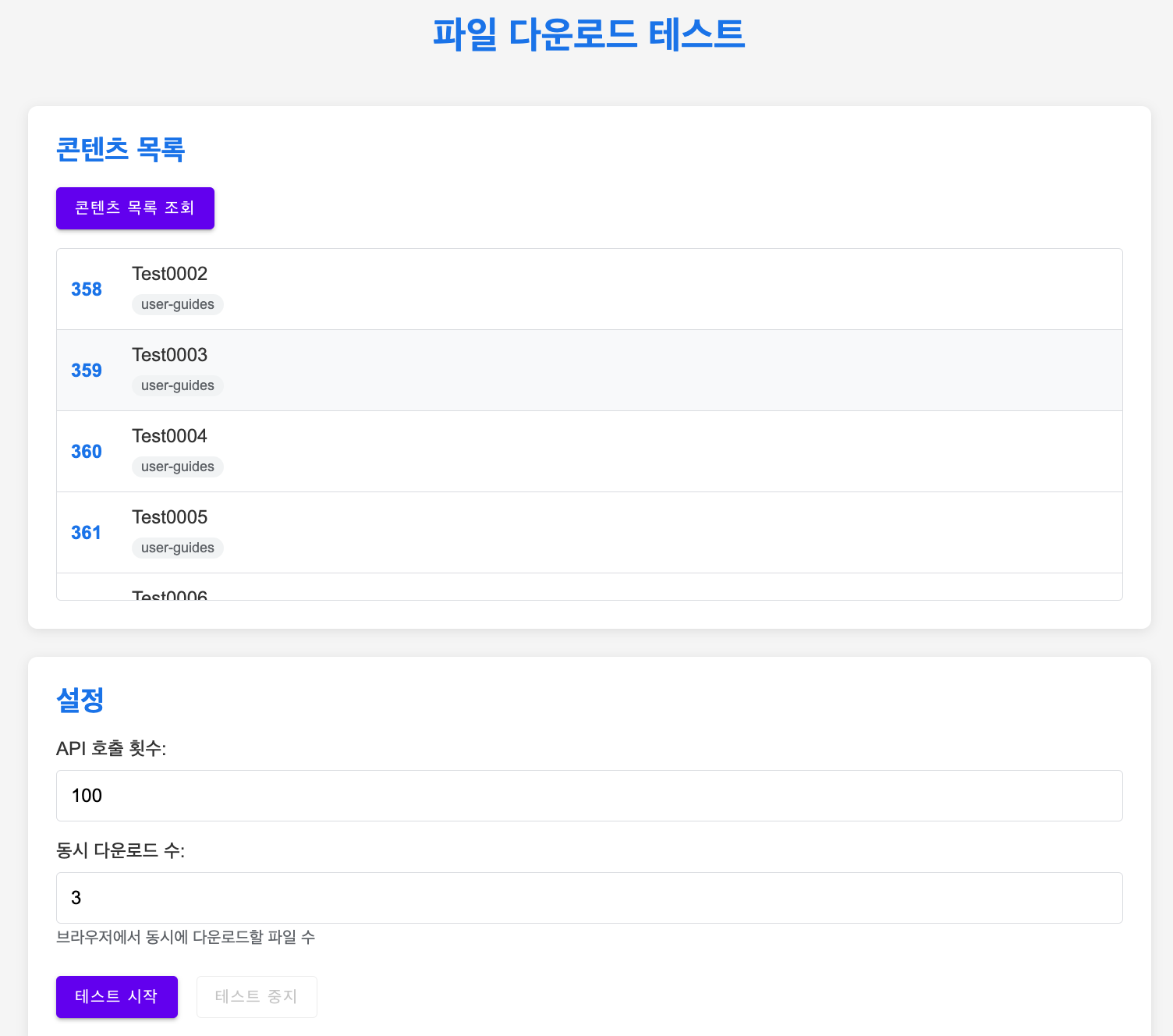
예)

| dd: 테스트 페이지를 개발해 주세요.  - 페이지 타이틀: 파일 다운로드  - 페이지 ID: download\_tester  - 포트: 3000  - FQDN: http://20.249.205.95.nip.io/api/contents/{filenum}  - Method: GET  - Request Header 'Authorization'에 아래값 추가  Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMDEiLCJyb2xlcyI6WyJVU0VSIl0sImlhdCI6MTc0MjQwMDkzMiwiZXhwIjoxNzQyNDA0NTMyfQ.-T9ZmhP1vkgMjhpWbmGCKuDy3iLZzKYE4phsV9lvhlI  - {filenum} 구하기  - http://20.249.205.95.nip.io/api/contents 호출 결과 이용  - 아래 예와 같은 응답에서 'contentId'이용  ```  [  {  "contentId": 358,  "contentName": "Test0002",  "contentType": "user-guides",  "description": "파일 업로드 테스트 - Test0002",  "thumbnailUrl": null  },  ]  ``` |
| --- |

**프롬프트 결과 예시: 부하테스트 스크립트 이후에 있음**

<https://claude.ai/share/101da694-1b1e-4600-a271-c2127f7c44b4>





※ QR코드 만들기

<https://qr.naver.com/>

