JIUCOM

MSA 인프라 매뉴얼

Docker + 모니터링 + API Gateway + 분산 추적

9개 서비스 아키텍처 동작 원리 가이드

문서명 : MSA 인프라 매뉴얼 v1.0

작성일 : 2026년 2월 27일

프로젝트 : JIUCOM API 플랫폼

기술 스택 : Spring Boot + Docker + Prometheus + Grafana

목차

1. 전체 아키텍처 개요

>> 아키텍처 설계 원칙

JIUCOM은 Docker Compose 기반의 마이크로서비스 지향 아키텍처로 총 9개의 서비스로 구성됩니다. 다음 4가지 핵심 원칙을 기반으로 설계되었습니다.

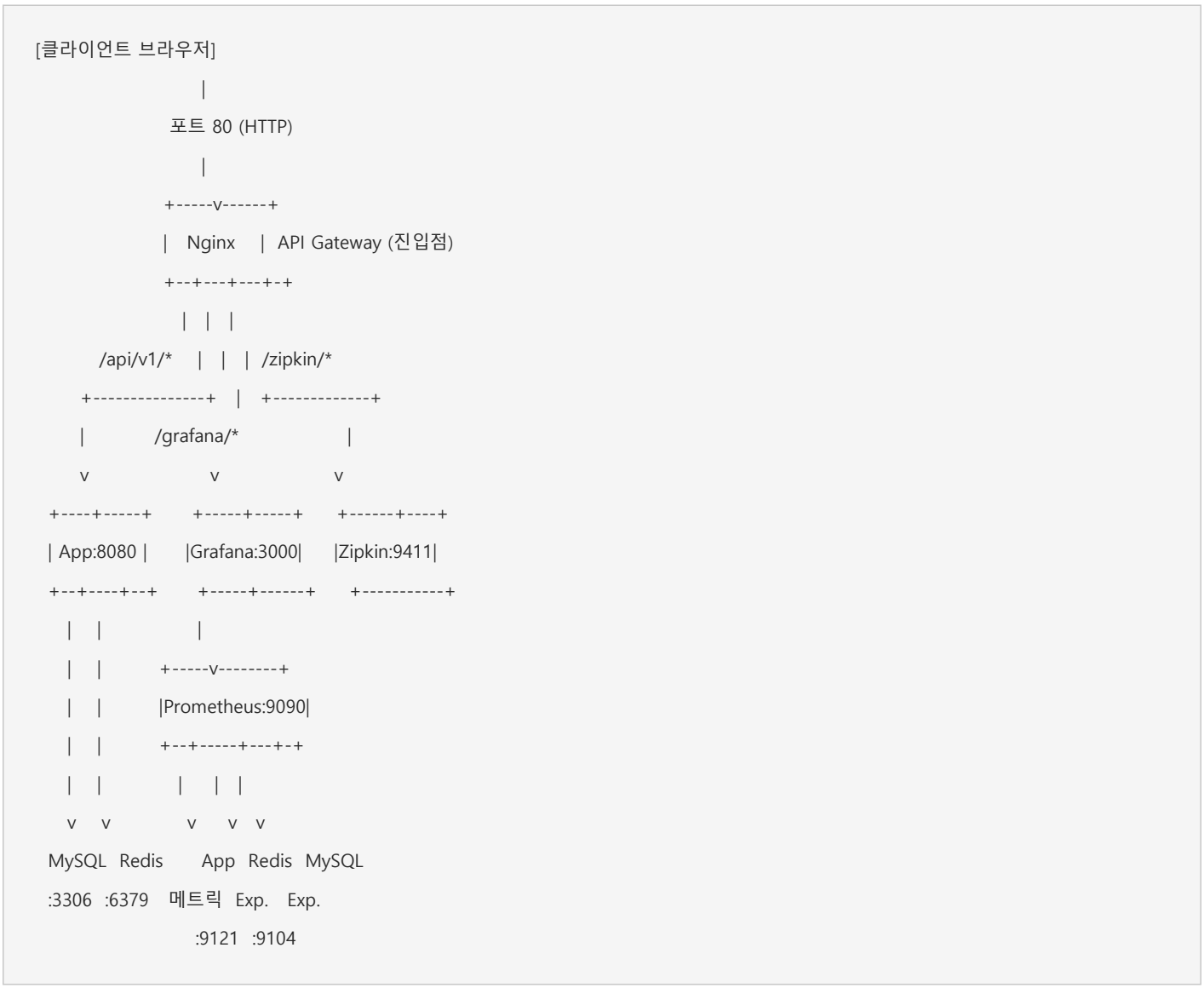
1) API Gateway 패턴 - Nginx가 모든 외부 트래픽의 단일 진입점 역할을 합니다. 요청 라우팅, 속도 제한(Rate Limiting), CORS, 보안 헤더 등 공통 관심사를 처리합니다.

2) 관측성(Observability) 스택 - Prometheus가 모든 서비스의 메트릭을 수집하고, Grafana가 시각화 대시보드를 제공하며, Zipkin이 분산 추적을 담당합니다.

3) 사이드카 익스포터 패턴 - Redis Exporter와 MySQL Exporter가 사이드카 컨테이너로 동작하여, 데이터베이스의 내부 지표를 Prometheus가 이해할 수 있는 형식으로 변환합니다.

4) 리소스 격리 - 모든 컨테이너에 CPU/메모리 제한을 설정하여 리소스 고갈을 방지하고, 프로덕션 수준의 인프라 환경을 시뮬레이션합니다.

>> 서비스 토폴로지 (전체 구조도)


[클라이언트 브라우저]
포트 80 (HTTP)
Nginx | API Gateway (진입점)
/api/v1/*
/zipkin/*
/grafana/*
App:8080
Grafana:3000
Zipkin:9411
Prometheus:9090
MySQL
Redis
App
Redis
MySQL
:3306
:6379
메트릭
Exp.
Exp.
:9121
:9104

2. 서비스 상세 설명

>> 9개 서비스 구성표

서비스	이미지	포트	역할	리소스
mysql	mysql:8.0	3306	메인 데이터베이스 (회원, 부품, 견적, 게시글 등)	2 CPU / 2GB
redis	redis:7-alpine	6379	캐시 + 세션 저장소 (가격 캐시, 속도 제한)	1 CPU / 512MB
app	Spring Boot (직접 빌드)	8080	JIUCOM API 서버 (70개 이상 REST 엔드포인트)	2 CPU / 1GB
nginx	nginx:alpine	80	API Gateway 리버스 프록시	0.5 CPU / 256MB
prometheus	prom/prometheus	9090	메트릭 수집 엔진 시계열 데이터베이스	0.5 CPU / 512MB
grafana	grafana/grafana	3000	모니터링 대시보드 시각화 도구	0.5 CPU / 512MB
redis-exp	oliver006/ redis_exporter	9121	Redis 메트릭 익스포터	0.25 CPU / 128MB
mysql-exp	prom/ mysqld-exporter	9104	MySQL 메트릭 익스포터	0.25 CPU / 128MB
zipkin	openzipkin/ zipkin	9411	분산 추적 서버 요청 흐름 추적	0.5 CPU / 512MB

>> 서비스 기동 순서 (의존성 체인)

Docker Compose는 depends_on 설정에 따라 서비스를 순서대로 기동합니다.

1단계) mysql, redis - 독립적으로 가장 먼저 시작 (헬스체크 완료까지 대기)
2단계) app - mysql과 redis가 정상(healthy) 상태가 된 후 시작
3단계) prometheus, zipkin - app이 시작된 후 메트릭 수집/추적 시작

4단계) grafana - prometheus가 시작된 후 대시보드 로드

5단계) redis-exporter, mysql-exporter - 각 DB가 정상인 후 메트릭 변환 시작

6단계) nginx - app, grafana, zipkin이 모두 준비된 후 마지막으로 시작

이 순서 덕분에 nginx가 시작될 때 모든 백엔드 서비스가 이미 준비 완료 상태입니다.

3. Nginx API Gateway 동작 원리

>> API Gateway란?

API Gateway는 마이크로서비스 아키텍처에서 모든 외부 요청이 거치는 단일 진입점입니다. 클라이언트는 각 서비스에 직접 접근하지 않고, 항상 Gateway를 통해 접근합니다.

JIUCOM에서 Nginx가 API Gateway 역할을 수행하며, 다음 기능을 담당합니다:

* 리버스 프록시 - URL 경로에 따라 적절한 백엔드 서비스로 요청을 전달합니다
* 속도 제한 (Rate Limiting) - IP당 초당 30회로 요청을 제한하여 DDoS/남용을 방지합니다
* 보안 헤더 - X-Frame-Options, X-Content-Type-Options, XSS 방어 헤더를 자동 추가합니다
* WebSocket 지원 - HTTP/1.1 Upgrade를 통해 실시간 알림(STOMP)을 지원합니다
* Gzip 압축 - JSON/텍스트 응답을 압축하여 전송 크기를 줄입니다

>> 라우팅 규칙

URL 경로	전달 대상	설명
/api/v1/*	app:8080	모든 REST API 엔드포인트 (인증, 부품, 견적, 게시글 등)
/grafana/*	grafana:3000	모니터링 대시보드 (Grafana 웹 UI)
/zipkin/*	zipkin:9411	분산 추적 UI (Zipkin 웹 콘솔)
/health	(직접 응답)	Nginx 자체 헬스체크 200 OK 반환
/	(리다이렉트)	Swagger UI로 자동 이동 /api/v1/swagger-ui.html

>> 리버스 프록시 동작 흐름

사용자가 http://localhost/api/v1/parts 를 요청하면 다음과 같이 동작합니다:

1) Nginx가 포트 80에서 요청을 수신합니다

2) 'location /api/v1/' 블록이 URL 경로와 매칭됩니다

3) Rate Limiter가 해당 클라이언트 IP의 요청 횟수를 확인합니다 (초당 30회 이내인지)

4) 허용되면, Nginx가 요청을 http://app:8080/api/v1/parts 로 전달(프록시)합니다

5) 이때 X-Real-IP, X-Forwarded-For, X-Forwarded-Proto 헤더를 추가합니다

6) Spring Boot 앱이 요청을 처리하고 응답을 반환합니다

7) Nginx가 보안 헤더를 추가한 후 클라이언트에게 응답을 전달합니다

핵심: 클라이언트는 Spring Boot 앱과 직접 통신하지 않습니다. Nginx가 중간에서 추상화 계층과 보안을 제공합니다.

4. 모니터링 스택 동작 원리

>> Prometheus - 메트릭 수집 엔진

Prometheus는 시계열 데이터베이스로, Pull 방식으로 모든 서비스의 메트릭을 수집합니다. 15초마다 각 대상(Target)의 메트릭 엔드포인트에 HTTP GET 요청을 보내 데이터를 가져옵니다.

Pull 방식: Prometheus --HTTP GET--> 대상 서비스 /metrics 엔드포인트

Pull 방식의 장점:
- 서비스가 Prometheus의 존재를 알 필요가 없습니다 (느슨한 결합)
- 새로운 서비스 추가 시 prometheus.yml에 대상만 추가하면 됩니다
- 서비스 장애 시에도 Prometheus가 독립적으로 동작합니다

prometheus.yml에 설정된 스크래핑 대상:

작업 이름	대상	메트릭 경로	수집하는 데이터
jiucom-api	app:8080	/api/v1/actuator/prometheus	JVM 힙 메모리, GC, 스레드 HTTP 요청 수, HikariCP 풀
redis-exporter	redis-exporter:9121	/metrics	연결된 클라이언트 수 메모리 사용량, 명령 실행 수
mysql-exporter	mysql-exporter:9104	/metrics	초당 쿼리 수, 스레드 수 슬로우 쿼리, 연결 수

>> Spring Boot가 메트릭을 노출하는 원리

Spring Boot 앱은 Micrometer(메트릭 파사드 라이브러리)를 사용하여 메트릭을 노출합니다.

동작 순서:
1) spring-boot-starter-actuator가 /actuator 엔드포인트를 활성화합니다
2) micrometer-registry-prometheus가 메트릭을 Prometheus 텍스트 형식으로 변환합니다
3) application.yml에서 노출할 엔드포인트를 설정합니다 (health, metrics, prometheus 등)
4) Prometheus가 15초마다 GET /api/v1/actuator/prometheus를 호출하여 수집합니다

주요 노출 메트릭:

* jvm_memory_used_bytes - JVM 힙/논힙 메모리 사용량
* jvm_threads_live_threads - 현재 활성 스레드 수
* http_server_requests_seconds_count - HTTP 요청 횟수 (상태코드/메서드/URI별)
* http_server_requests_seconds_sum - 총 응답 시간 (평균 계산용)
* hikaricp_connections_active - 현재 사용 중인 DB 커넥션 수
* jvm_gc_pause_seconds_count - 가비지 컬렉션 발생 횟수

>> Grafana - 시각화 대시보드

Grafana는 Prometheus에서 데이터를 읽어 시각적 대시보드로 표시하는 도구입니다. JIUCOM 대시보드는 컨테이너 시작 시 자동으로 프로비저닝(설정)됩니다.

자동 프로비저닝 파일 (Docker 볼륨으로 마운트):

* datasource.yml - Grafana를 Prometheus에 연결 (http://prometheus:9090)
* dashboard.yml - /var/lib/grafana/dashboards/ 경로에서 대시보드를 자동 로드하도록 설정
* jiucom-dashboard.json - 미리 만들어진 16개 패널의 대시보드 정의 파일

자동 프로비저닝 동작 원리:
1) Grafana 컨테이너가 시작됩니다
2) /etc/grafana/provisioning/ 디렉토리의 설정 파일을 읽습니다
3) datasource.yml을 읽고 Prometheus 데이터소스를 자동 등록합니다
4) dashboard.yml을 읽고 대시보드 JSON 파일의 위치를 확인합니다
5) jiucom-dashboard.json을 로드하여 대시보드를 자동 생성합니다
6) 별도의 수동 설정 없이 즉시 대시보드를 사용할 수 있습니다

>> 익스포터(Exporter) 패턴

Redis와 MySQL은 자체적으로 Prometheus 메트릭을 노출하지 않습니다. 그래서 익스포터라는 사이드카 컨테이너가 중간에서 번역 역할을 합니다.

동작 원리:
1) 익스포터가 데이터베이스(Redis/MySQL)에 네이티브 프로토콜로 접속합니다
2) 내부 통계를 조회합니다 (Redis: INFO 명령, MySQL: SHOW GLOBAL STATUS)
3) 조회한 통계를 Prometheus가 이해할 수 있는 메트릭 형식으로 변환합니다
4) /metrics 엔드포인트로 노출하여 Prometheus가 스크래핑할 수 있게 합니다

이것은 Prometheus 생태계의 표준 패턴으로, 자체 메트릭을 노출하지 않는 서드파티 시스템을 모니터링할 때 사용합니다.

5. 분산 추적 (Zipkin) 동작 원리

>> 분산 추적이란?

마이크로서비스 아키텍처에서는 하나의 사용자 요청이 여러 서비스를 거쳐 처리됩니다. 분산 추적은 이 요청이 어떤 서비스들을 거쳤는지, 각 서비스에서 얼마나 시간이 걸렸는지, 어디서 병목이나 오류가 발생했는지를 추적하는 기술입니다.

핵심 개념:
- Trace ID: 하나의 요청 전체를 식별하는 고유 ID
- Span ID: 요청 내 각 개별 작업(서비스 호출)을 식별하는 ID
- 하나의 Trace는 여러 개의 Span으로 구성됩니다

>> JIUCOM에서의 동작 흐름

1) 클라이언트가 Nginx를 통해 Spring Boot 앱에 요청을 보냅니다

2) Micrometer Tracing (Brave 브릿지)이 자동으로 다음을 수행합니다:
   - 요청에 대한 고유 Trace ID를 생성합니다
   - HTTP 핸들러, DB 쿼리, Redis 호출 등 각 작업에 Span을 생성합니다
   - B3 전파 방식으로 HTTP 헤더를 통해 추적 컨텍스트를 전달합니다

3) 요청 처리가 완료되면, Span 데이터가 비동기적으로 Zipkin에 전송됩니다
   (HTTP POST -> http://zipkin:9411/api/v2/spans)

4) Zipkin이 추적 데이터를 저장합니다 (개발: 인메모리, 프로덕션: Elasticsearch 가능)

5) Zipkin UI (포트 9411)에서 타임라인/워터폴 다이어그램으로 추적 결과를 확인합니다

>> 추가된 의존성

```
// build.gradle.kts - 분산 추적 라이브러리
implementation("io.micrometer:micrometer-tracing-bridge-brave")
implementation("io.zipkin.reporter2:zipkin-reporter-brave")

// application.yml - 추적 설정
management.tracing.sampling.probability: 1.0     # 샘플링 비율
management.zipkin.tracing.endpoint: http://zipkin:9411/api/v2/spans
```

>> 샘플링 전략

프로필	샘플링 비율	이유
dev (개발)	1.0 (100%)	모든 요청을 추적하여 디버깅에 활용
prod (운영)	0.1 (10%)	성능 오버헤드를 줄이면서 충분한 분석 데이터 확보

왜 운영 환경에서 100%가 아닌 10%만 샘플링하는가?

- 모든 요청을 추적하면 네트워크 전송과 저장 비용이 과도해집니다
- 10%만 샘플링해도 통계적으로 충분한 패턴 분석이 가능합니다
- 문제 발생 시 일시적으로 100%로 올려 디버깅할 수 있습니다

6. Docker Compose 오케스트레이션

>> 컨테이너 리소스 제한

모든 컨테이너에 CPU와 메모리 제한을 설정하여 하나의 서비스가 시스템 자원을 독점하는 것을 방지합니다. 이는 Kubernetes의 resources.limits/requests와 동일한 개념입니다.

서비스	CPU 최대	메모리 최대	CPU 최소 보장	메모리 최소 보장
app	2.0 코어	1 GB	0.5 코어	512 MB
mysql	2.0 코어	2 GB	0.5 코어	512 MB
redis	1.0 코어	512 MB	0.25 코어	128 MB
nginx	0.5 코어	256 MB	0.1 코어	64 MB
prometheus	0.5 코어	512 MB	0.25 코어	256 MB
grafana	0.5 코어	512 MB	0.25 코어	256 MB
zipkin	0.5 코어	512 MB	0.25 코어	256 MB
redis-exp	0.25 코어	128 MB	0.1 코어	64 MB
mysql-exp	0.25 코어	128 MB	0.1 코어	64 MB

리소스 제한의 의미:
- limits (최대): 컨테이너가 절대 초과할 수 없는 상한선. 초과 시 OOM Kill됩니다
- reservations (최소 보장): 컨테이너에 최소한 보장되는 자원. 다른 컨테이너가 아무리 바빠도 이 만큼의 자원은 항상 사용 가능합니다

>> 네트워크 아키텍처

모든 서비스는 하나의 Docker 브릿지 네트워크(jiucom-network)를 공유합니다.

이 네트워크 안에서 컨테이너들은 서비스 이름을 DNS 호스트명으로 사용하여 통신합니다. 예를 들어 app 컨테이너는 'mysql:3306'과 'redis:6379'로 접속합니다. Docker의 내장 DNS가 이 이름을 컨테이너 IP 주소로 자동 변환합니다.

외부에 공개되는 포트:
- 포트 80 (Nginx) - 메인 진입점
- 포트 8080 (App) - 직접 API 접근 (개발용)
- 포트 3000 (Grafana), 9090 (Prometheus), 9411 (Zipkin) - 모니터링

>> 볼륨 영속성

볼륨 이름	컨테이너 경로	용도
mysql-data	/var/lib/mysql	MySQL 데이터 파일 (컨테이너 재시작 시에도 데이터 유지)
redis-data	/data	Redis RDB 스냅샷 (캐시 데이터 영속화)
prometheus-data	/prometheus	시계열 메트릭 데이터 (15일간 보관)
grafana-data	/var/lib/grafana	Grafana 설정, 사용자 정보 (대시보드는 프로비저닝으로 별도 관리)

7. 데이터 흐름 - 요청의 전체 생명주기

>> 예시: GET /api/v1/parts (부품 검색)

사용자가 컴퓨터 부품을 검색할 때의 단계별 흐름입니다:

1단계: 요청 진입 (Request Ingress)

* 사용자가 HTTP GET http://localhost/api/v1/parts?category=CPU 요청을 보냅니다
* Nginx가 포트 80에서 수신하고, 'location /api/v1/' 규칙과 매칭합니다
* Rate Limiter가 확인합니다: 이 IP가 초당 30회 이내인가? 맞으면 통과
* Nginx가 http://app:8080/api/v1/parts?category=CPU 로 요청을 전달합니다
* X-Real-IP, X-Forwarded-For, X-Forwarded-Proto 헤더를 추가합니다

2단계: 애플리케이션 처리 (Application Processing)

* Spring Boot가 요청을 수신하고, Micrometer가 Trace ID + Span을 생성합니다
* JwtAuthenticationFilter가 Authorization 헤더를 확인합니다 (있는 경우)
* RequestLoggingFilter가 요청 정보를 로깅합니다 (메서드, URI, 클라이언트 IP)
* DispatcherServlet이 PartController.searchParts() 메서드를 호출합니다
* PartService가 PartRepository를 통해 QueryDSL 동적 쿼리를 실행합니다
* HikariCP가 커넥션 풀에서 DB 커넥션을 하나 할당합니다
* MySQL에서 쿼리가 실행되고, 결과가 PartSearchResponse DTO로 매핑됩니다

3단계: 응답 반환 (Response Egress)

* ApiResponse<T>가 결과를 감쌉니다: {success: true, data: [...], message: ...}
* Micrometer가 메트릭을 기록합니다: http_server_requests_seconds (소요시간, 상태코드)
* 추적 Span이 비동기로 Zipkin에 전송됩니다 (http://zipkin:9411/api/v2/spans)
* 응답이 Nginx를 거쳐 반환되며, 보안 헤더가 추가됩니다
* 클라이언트가 부품 데이터가 담긴 JSON 응답을 수신합니다

4단계: 메트릭 수집 (백그라운드에서 상시 동작)

* 15초마다 Prometheus가 GET /api/v1/actuator/prometheus를 호출합니다
* 수집 항목: 요청 횟수, 응답 시간, JVM 힙 메모리, 스레드 수 등
* Prometheus가 TSDB(시계열 데이터베이스)에 저장합니다 (15일간 보관)
* Grafana가 PromQL로 Prometheus를 조회하여 대시보드에 시각화합니다
* Redis Exporter는 Redis INFO를, MySQL Exporter는 SHOW STATUS를 스크래핑합니다

>> 모니터링 데이터 흐름도

8. 대시보드 설계와 모니터링 지표

>> Grafana 대시보드 레이아웃

사전 구성된 JIUCOM 대시보드 (UID: jiucom-main)는 4개 행 섹션에 총 16개 패널로 구성됩니다. 10초마다 자동 새로고침됩니다.

행	패널 유형	모니터링 대상	경고 신호
개요 (1행)	Stat 패널 x5	API 상태 (UP/DOWN)	DOWN 상태
		힙 사용률 (%)	힙 90% 초과
		HTTP 요청 속도	속도 급증/급감
HTTP (2행)	시계열 x2	평균 응답 시간	지연 시간 > 1초
		상태코드별 요청 수	5xx 오류 > 1%
JVM (3행)	시계열 x2	힙 메모리 추이	메모리 지속 증가
		GC 일시정지 빈도	잦은 Full GC
DB/캐시 (4행)	시계열 x5	HikariCP 풀 사용량	풀 소진
		Redis 클라이언트/메모리	Redis 메모리 포화
		MySQL 쿼리/스레드	슬로우 쿼리 급증

>> 주요 PromQL 쿼리 (Grafana에서 사용)

```
# API 서버 가동 상태 (1=UP, 0=DOWN)
up{job="jiucom-api"}

# JVM 힙 메모리 사용률 (%)
jvm_memory_used_bytes{area="heap"} / jvm_memory_max_bytes{area="heap"} * 100

# HTTP 초당 요청 수 (5분 평균)
rate(http_server_requests_seconds_count{job="jiucom-api"}[5m])

# HTTP 5xx 오류 비율 (%)
rate(http_server_requests_seconds_count{status=~"5.."}[5m])
  / rate(http_server_requests_seconds_count[5m]) * 100

# 평균 응답 시간 (초)
rate(http_server_requests_seconds_sum[5m])
  / rate(http_server_requests_seconds_count[5m])
```

PromQL 읽는 법:

- rate(...[5m]): 최근 5분간의 초당 변화율을 계산합니다
- {job="jiucom-api"}: jiucom-api 작업에서 수집된 메트릭만 필터링합니다
- {status=~"5.."}: 정규식으로 500~599 상태코드를 필터링합니다
- sum/count 나누기: 총 소요시간 / 총 요청 수 = 평균 응답 시간

9. 운영 가이드

>> 시작 / 종료 명령어

```
# 전체 9개 서비스 시작 (앱 이미지 빌드 포함)
docker compose up -d --build

# 재빌드 없이 시작
docker compose up -d

# 전체 서비스 종료 (데이터 볼륨은 유지)
docker compose down

# 전체 종료 + 모든 데이터 삭제 (주의!)
docker compose down -v

# 특정 서비스 로그 확인
docker compose logs -f app
docker compose logs -f nginx

# 특정 서비스만 재시작
docker compose restart app
```

>> 상태 확인 URL

서비스	URL	정상 응답
API (Nginx 경유)	http://localhost/api/v1/actuator/health	200 OK {status: UP}
API (직접 접근)	http://localhost:8080/api/v1/actuator/health	200 OK {status: UP}
Prometheus	http://localhost:9090/targets	3개 타겟 모두 UP
Grafana	http://localhost:3000/grafana/login	로그인 페이지 표시 (admin/admin)

Zipkin	http://localhost:9411/zipkin/	Zipkin UI 표시
Swagger	http://localhost/api/v1/swagger-ui.html	API 문서 페이지

>> 문제 해결 가이드

문제 상황	진단 방법	해결 방법
앱이 시작되지 않음	docker compose logs app	mysql 헬스 상태 확인
	DB 연결 확인	DB_HOST 환경변수 점검
Prometheus	http://localhost:9090	prometheus.yml 확인
타겟 없음	/targets 접속	앱 실행 여부 점검
Grafana	Grafana 데이터소스	Prometheus URL 확인
데이터 없음	설정 확인	http://prometheus:9090
Nginx	docker compose logs nginx	app 컨테이너가
502 Bad Gateway	업스트림 상태 확인	실행 중인지 확인
Zipkin에	앱 로그에서 Zipkin	ZIPKIN_ENDPOINT
추적 없음	리포터 오류 확인	환경변수 점검
MySQL 익스포터	docker compose logs	DATA_SOURCE_NAME 확인
연결 거부	mysql-exporter	MySQL 헬스 상태 점검

>> 프로덕션 확장 시 고려사항

실제 서비스 배포 시 다음 업그레이드를 고려해야 합니다:

1) Nginx: Let's Encrypt 인증서로 SSL/TLS 종료(HTTPS) 설정
2) App: 'docker compose up --scale app=3'으로 수평 확장 (로드밸런싱)
3) Zipkin: 인메모리에서 Elasticsearch로 저장소 변경 (데이터 영속화)
4) Prometheus: Alertmanager 추가하여 Slack/이메일 알림 자동 발송
5) Grafana: SMTP 설정으로 정기 리포트 이메일 발송
6) MySQL: 읽기 전용 복제본(Read Replica) 추가로 쿼리 분산
7) Redis: Redis Cluster 모드로 고가용성 확보

JIUCOM - 프로덕션 수준의 MSA 인프라 아키텍처

문서 생성일: 2026-02-27 | JIUCOM Team