

게이트웨이 안내서

AnyAPI 1.0

TMAXSOFT

저작권 공지

Copyright 2025. TmaxSoft Co., Ltd. All Rights Reserved.

회사 정보

(주)티맥스소프트

주소 : 경기도 성남시 분당구 정자일로 45, 티맥스소프트타워

기술 서비스 센터: 1544-8629

홈페이지: <https://www.tmaxsoft.com>

제한된 권리

이 소프트웨어(AnyAPI™) 사용설명서와 프로그램은 저작권법과 국제 조약에 의해 보호됩니다. 사용설명서와 프로그램은 TmaxSoft Co., Ltd.와의 사용권 계약 하에서만 사용할 수 있으며, 사용설명서는 사용권 계약의 범위 내에서만 배포 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부를 TmaxSoft의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단으로 전송, 복제, 배포하거나 2차적 저작물을 작성할 수 없습니다.

이 소프트웨어 사용설명서와 프로그램의 사용권 계약은 어떠한 경우에도 사용설명서 및 프로그램과 관련된 지적 재산권(등록 여부를 불문)을 양도하는 것으로 해석되지 않으며, 브랜드나 로고, 상표 등을 사용할 권한을 부여하지 않습니다. 사용설명서는 오로지 정보 제공만을 목적으로 하며, 이로 인한 계약상의 직접적 또는 간접적 책임을 지지 않습니다. 또한 사용설명서 상의 내용이 법적 또는 상업적인 특정 조건을 만족시킬 것을 보장하지 않습니다. 사용설명서는 제품의 업그레이드나 수정에 따라 예고 없이 변경될 수 있으며, 내용상의 오류가 없음을 보장하지 않습니다.

상표 공지

AnyAPI™는 TmaxSoft Co., Ltd.의 상표입니다. 본 사용설명서에 기재된 모든 제품과 회사 이름은 각각 해당 소유주의 상표로서 참조용으로만 사용되며 반드시 상표 표시(™, ®)를 하지는 않습니다.

Noto는 Google Inc.의 상표입니다. Noto 글꼴은 오픈 소스입니다. 모든 Noto 글꼴은 SIL Open Font License, 버전 1.1에 따라 게시됩니다. (<https://www.google.com/get/noto/>)

오픈소스 소프트웨어 공지

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다. : MIT, LGPL, PSFL

관련 상세 정보는 제품의 다음 디렉터리에 기재된 사항을 참고하시기 바랍니다. : \${INSTALL_PATH}/lib/licenses

유지 보수

구분	지원항목	서비스 내용
제품지원	패치 & 업그레이드	무상 패치 서비스 제공 메이저 버전 업그레이드 시 할인 혜택 웹 지원을 통한 패치 내역 제공
기술 지원 - 기본 서비스	장애 지원	장애 발생 시 원인 분석 및 조치 Service Desk팀 → 기술팀 → R&D의 3단계 장애 분석 및 조치
	일상 지원(온라인 지원)	E-mail, 전화, 원격, 웹 사이트 등 온라인 자원을 통한 질의 응답 서비스
	고객 맞춤 지원(방문 지원)	고객의 요청으로 수행하는 방문 지원 서비스
기술 지원 - 옵션 서비스	예방 지원	정기 점검을 통한 시스템 운영현황 보고 및 장애 예방 ◦ 관리자 또는 운영자의 요구사항 수렴 ◦ 운영 현황(시스템, 엔진 운영) 보고서 제공 ◦ 필요 시 시스템 개선 권장 사항 보고
유지 보수 비용 및 기간	계약 시 별도 협의	계약 시 EOL/EOS 문서 제공

안내서 이력

제품 버전	안내서 버전	발행일	비고
AnyAPI 1.0	3.1.4	2025-09-10	시스템 요구 사항 최신화
AnyAPI 1.0	3.1.3	2025-05-29	신규 기능 추가 ◦ HMAC 인증 기능 ◦ Aggregation API 관리 기능
AnyAPI 1.0	3.1.2	2025-01-02	-
AnyAPI 1.0	3.1.1	2024-08-30	-

목차

1. 소개	1
1.1. 개요	1
1.2. 특징	1
1.3. 게이트웨이 환경	2
1.3.1. 게이트웨이 설정 파일	2
1.3.2. 디렉터리 구조	12
1.3.3. 환경 변수	13
2. 엔드포인트 구성	15
2.1. 개요	15
2.1.1. 주요 특징	15
2.1.2. AnyAPI 기본 엔드포인트	15
2.2. 엔드포인트 설정	16
2.2.1. SSL Passthrough 설정	18
3. 모니터링	20
3.1. 모니터링 기본 개념	20
3.1.1. 메트릭(Metric)	20
3.1.2. 트레이스(Trace)	21
3.1.3. gRPC	21
3.1.4. OpenTelemetry	21
3.2. 모니터링 구조 및 흐름	22
3.2.1. 모니터링 전체 구조	22
3.2.2. 모니터링 데이터 수집 과정	22
3.2.3. 모니터링 데이터 전달 과정	24
3.2.4. 모니터링 서버 처리 과정	25
3.3. 게이트웨이 설정	25
3.3.1. 트레이스 설정 예시	25
3.3.2. 메트릭 설정 예시	27
3.4. 모니터링 Kafka 연동	28
3.4.1. 구성 및 동작 방식	28
3.4.2. 설정 방법	29
부록 A: 문제 해결	34
A.1. FailedGatewayReadsizeBelowZero	34
A.2. GatewayAPIAuthenticationFailException	34
A.3. GatewayAPINoMatchingConditionException	34
A.4. GatewayEndpointNotExistException	34
A.5. GatewaySslInitFailException	34
A.6. SSL unknown error	35
A.7. Invalid HTTP method	35

1. 소개

본 장에서는 AnyAPI 게이트웨이의 특징 및 설정 파일의 설정 방법에 대해 설명합니다.

1.1. 개요

AnyAPI 게이트웨이는 클라이언트의 API 요청을 AnyAPI 게이트웨이에 등록된 라우팅 정보에 따라 각 API 서버들로 전달하고, 각 API 서버들로부터 받은 응답들을 취합하여 클라이언트에 전달합니다.

또한 라우팅 정보를 통하여 요청 URI 검사, 파라미터 검사 등을 수행합니다. 여러 검사 결과에 따라 라우팅할 Backend API 서버의 URI가 특정되면 해당 정보를 마스터로 전달하여 인증/인가, Qos 처리를 추가로 진행합니다. 이후 다시 게이트웨이를 통하여 실제 API 서버로 라우팅을 한 후 응답을 돌려 받습니다.

1.2. 특징

AnyAPI 게이트웨이의 특징은 다음과 같습니다.

- **Non-Blocking 방식**

I/O 멀티플렉싱 방식의 입출력 처리를 하며, 특정 I/O 채널의 버퍼링에 다른 채널 처리가 영향받지 않도록 fully non-blocking 방식의 처리를 구현하고 있습니다. 기본적으로 1개의 프로세스로 이루어져 하나의 이벤트 스레드에서 non blocking I/O를 통해 HTTP 프로토콜을 처리하며 데이터에 대한 암호화 처리, blocking I/O가 필요한 경우 워커 스레드를 통해 처리합니다.

- **HTTP 프로토콜 지원**

HTTP 버전으로 1.0, 1.1 두 종류를 지원하며, HTTPS 프로토콜을 지원합니다. 또한 HTTP로 들어온 요청을 HTTPS로 변환하는 기능도 지원합니다.

- **라우팅 및 변환**

HTTP 요청이 들어오면 Path나 Method를 기반으로 API를 식별하는 라우팅을 지원합니다. Path에서 와일드카드(*)를 사용하여 모든 문자열에 대응할 수 있습니다. 또한 Path나 Method를 다른 Path, Method로 변환을 지원합니다. Path의 경우 특정 예약어를 통해 지정한 문자열로 대체할 수 있습니다.

항목	설명
{FullPath}	들어온 요청의 전체 Path로 치환합니다.
{StagePath}	Stage의 basePath로 치환합니다.
{Path}	들어온 요청의 전체 Path에서 stage basePath를 제외한 Api Path로 치환합니다.

- **목적지 설정**

서버나 서버 그룹 또는 IP를 직접 지정하여 해당 경로로 프록시 또는 리다이렉트 응답을 돌려주는 기능을 지원합니다. 서버 그룹을 설정하여 라운드 robin 방식, Weight 방식의 목적지 라우팅을 설정할 수 있습니다.

• API 배포 및 테스트 기능

구현한 API는 배포 버튼을 누르기 전까지 외부에 노출되지 않습니다. 배포하지 않은 API는 AnyAPI의 테스트 기능을 통해 정상적으로 동작하는지, API 서버가 어떤 응답을 돌려주는지, API 서버로부터 응답이 얼마나 걸리는지를 확인할 수 있습니다.

• 인증/인가

인증된 사용자를 확인하는 인증 기능과 API 호출 횟수를 제한할 수 있는 OQS 인가 기능을 제공합니다. QOS는 STAGE별로 설정이 가능하며 분당, 시간당, 하루당 얼마큼 처리할 것인지 지정할 수 있습니다.

다음은 AnyAPI 게이트웨이에서 현재 제공하는 인증 방식의 종류입니다.

구분	설명
API Key 인증	HTTP 헤더에 들어온 x-api-key 헤더를 확인하여 등록된 API Key인지 확인하여 사용자를 인증하는 방식입니다.
OAuth 인증	OAuth 서버로부터 접근 토큰을 발급받아 Authorization 헤더에 넣어서 요청하면 AnyAPI 게이트웨이는 Authorization 토큰이 유효한지 체크하여 사용자를 인증하는 방식입니다.
Terminal 인증	TerminalLogin API를 호출하여 단말 토큰을 생성한 후 x-api-token, x-api-pk-number 헤더에 넣어서 요청하면 AnyAPI 게이트웨이는 해당 토큰이 유효한지 체크하여 사용자를 인증하는 방식입니다.

1.3. 게이트웨이 환경

1.3.1. 게이트웨이 설정 파일

AnyAPI 게이트웨이에서 사용하는 설정 파일은 초기 게이트웨이 설정과 동적으로 반영되는 게이트웨이 설정을 저장하기 위해 사용합니다.

설치 초기에 설정 파일은 다음과 같은 위치에 존재합니다.

```
`${GATEWAY_HOME}/config/defaultApiGateway21
```

다음은 AnyAPI 게이트웨이 설정 파일의 작성 예시입니다. (현재 JSON 형식만 지원)



IP, PORT, Monitoring 설정을 제외하고는 사용자가 직접 수정할 수 없으며, WebAdmin을 통해서만 수정이 가능합니다.

• gatewayConfig

게이트웨이에 대한 기본 메타 데이터 설정 정보입니다.

```

"gatewayConfig": {
  "gatewayGroupId": "${GROUP_ID}", (1)
  "id": "${ID}", (2)
  "name": "${NAME}", (3)
  "protocol": "HTTP" (4)
  "thread_monitoring": 1000 (5)
  "thread_pool": { (6)
    "min": 10,
    "max": 50
  }
}

```

각 설정 항목에 대한 설명은 다음과 같습니다.

항목	설명
(1) gatewayGroupId	게이트웨이가 배포되는 그룹의 ID입니다.
(2) id	게이트웨이의 고유 ID입니다.
(3) name	게이트웨이의 이름입니다. 설정하지 않는 경우 ID를 사용합니다.
(4) protocol	게이트웨이가 사용하는 프로토콜입니다. (기본값: HTTP)
(5) thread_monitoring	스레드 모니터링용 로그의 주기(ms)를 설정합니다. (기본값: 1000)
(6) thread_pool	<p>기본 스레드 풀을 설정합니다.</p> <p>실제 파싱이나 매핑이 이루어질 때 사용됩니다. 이때 스레드 풀은 선형적으로 증가하지는 않으며, 상황에 맞게 max 값까지 증가합니다.</p> <ul style="list-style-type: none"> ◦ min: 부팅 시 최소로 뜨는 스레드 개수 ◦ max: 최대로 늘어나는 스레드 개수

• staticResources

게이트웨이가 처음 부팅될 때 필요한 설정 정보입니다. 이때 "endpoints" 필드는 외부와의 통신 정보이고, "applications" 필드는 게이트웨이에 등록된 애플리케이션의 정보입니다.

```

"staticResources": {
  "endpoints": [{
    "endpoint": {
      "physicalName": "poClientEndpoint",
      "protocol": "HTTP",
      "direction": "OUTBOUND",
      "bootState": "RUNNING",
      "connectType": "CLIENT",
      "httpUrl": {
        "split": true,
        "scheme": "http",
        "host": "${MS_IP}",
        "port": "${MS_PORT}"
      }
    }
  ]
},

```

```

{
  "endpoint": {
    "physicalName": "poServerEndpoint",
    "protocol": "HTTP",
    "bootState": "RUNNING",
    "connectionPool": {
      "max": "1000"
    },
    "httpUrl": {
      "split": true,
      "scheme": "http",
      "host": "${GW_IP}",
      "port": "${GW_PORT}"
    },
    "rulesetId": "anylink.system.SystemServiceRuleSet"
  }
},
{
  "endpoint": {
    "physicalName": "httpEndpoint",
    "protocol": "HTTP",
    "bootState": "Running",
    "connectType": "Server",
    "connection_pool": {
      "max": "100"
    },
    "httpUrl": {
      "scheme": "http",
      "port": "${HTTP_PORT}"
    },
    "rulesetId": "ApiApplication.ApiServiceGroup.ApiDefaultRuleChain",
    "errorRulesetId": "anylink.system.ApiGatewayErrorRuleSet"
  }
},
{
  "endpoint": {
    "physicalName": "httpsEndpoint",
    "protocol": "HTTP",
    "bootState": "Running",
    "connectType": "Server",
    "connectionPool": {
      "max": "100"
    },
    "useSsl": true,
    "ssl": {
      "keystore": {
        "storeType": "PEM",
        "storeLocation": "${SSL_STORE}"
      }
    },
    "httpUrl": {
      "scheme": "http",
      "port": "${HTTPS_PORT}"
    },
    "rulesetId": "ApiApplication.ApiServiceGroup.ApiDefaultRuleChain",
    "errorRulesetId": "anylink.system.ApiGatewayErrorRuleSet"
  }
}
}],
"applications": [{

```



```

    "id": <string>, (1)
    "version": <int>, (2)
    "servicegroups": [{ (3)
        "id": <string>, (4)
        "version": <int>, (5)
        "rulesets": [{ (6)
            "id": <string>, (7)
            "rule_chains": <RuleChainInfo> (8)
        }],
        "libpath": <string> (9)
    }]
}

```

각 설정 항목에 대한 설명은 다음과 같습니다.

항목	설명
(1) id	애플리케이션의 ID입니다.
(2) version	애플리케이션의 버전입니다.
(3) servicegroups	애플리케이션의 하위 서비스 그룹입니다.
(4) id	서비스 그룹의 ID입니다.
(5) version	서비스 그룹의 버전입니다.
(6) rulesets	서비스 그룹의 룰셋입니다.
(7) id	룰셋의 ID입니다.
(8) rule_chains	체인으로 정의한 룰셋의 실제 내용입니다.
(9) libpath	읽어들일 SharedLibrary의 경로입니다.



"endpoints" 필드의 설정 방법에 대한 자세한 내용은 [엔드포인트 설정](#)을 참고합니다.

• dynamicResources

게이트웨이가 웹 어드민으로부터 받아온 동적 설정 정보입니다. 해당 필드는 부팅 시 자동으로 설정되며, 웹 어드민을 통해서만 수정이 가능합니다.

```

"dynamicResources": {
    "string": <int>
    "endpoints": [<EndpointType>], (1)
    "applications": [<ApplicationType>], (2)
}

```

각 설정 항목에 대한 설명은 다음과 같습니다.

항목	설명
(1) endpoints	엔드포인트 목록입니다.

항목	설명
(2) applications	애플리케이션 목록입니다.

• EndpointInfo

게이트웨이에 등록된 endpoint 설정 정보입니다. 해당 필드는 웹 어드민을 통해서만 수정이 가능합니다.

```
"EndpointInfo": {
  "logical_name": <string>, (1)
  "physical_name": <string>, (2)
  "protocol": ("PO", "HTTP", "TCP"), (3)
  "direction": ("INBOUND", "OUTBOUND", "BOTH"),
  "boot_state": ("STOPPED", "RUNNING"),
  "http_url": { (4)
    "split": <bool>, (5)
    "url": <string>, (6)
    "scheme": ("HTTP", "HTTPS"), (7)
    "host": <string>, (8)
    "port": <string> (9)
  }
}
```

각 설정 항목에 대한 설명은 다음과 같습니다.

항목	설명
(1) logical_name	엔드포인트의 논리적인 이름입니다.
(2) physical_name	엔드포인트의 ID입니다.
(3) protocol	엔드포인트의 프로토콜입니다. (현재 HTTP만 사용 가능)
(4) http_url	연결할 원격 HTTP 서버의 URL입니다.
(5) split	URL의 분리 여부입니다. <ul style="list-style-type: none"> ◦ true: URL을 분리함 ◦ false: URL을 분리 안 함
(6) url	전체 URL입니다. 단, split이 'false'일 때 설정합니다.
(7) scheme	URL Scheme입니다. 단, split이 'true'일 때 설정합니다. <ul style="list-style-type: none"> ◦ HTTP ◦ HTTPS
(8) host	HTTP 호스트 이름입니다. 단, split이 'true'일 때 설정합니다.
(9) port	HTTP 포트 번호입니다. 단, split이 'true'일 때 설정합니다.

• EndpointGroupInfo

게이트웨이에 등록된 endpointgroup 설정 정보입니다. 해당 필드는 웹 어드민을 통해서만 수정이 가능합니다.

```
"EndpointGroupInfo":{
  "logical_name": <string>, (1)
  "physical_name": <string>, (2)
  "routing_type": ("ROUND_ROBIN", "WEIGHT_BASED", "PRIORITY"), (3)
  "routing": [{
    "endpoint_id": <string>, (4)
    "priority": <int>, (5)
    "weight": <int> (6)
  }]
}
```

각 설정 항목에 대한 설명은 다음과 같습니다.

항목	설명
(1) logical_name	엔드포인트의 논리적인 이름입니다.
(2) physical_name	엔드포인트의 ID입니다.
(3) routing_type	엔드포인트의 라우팅 정책입니다. (현재 WEIGHT_BASED만 지원)
(4) endpoint_id	라우팅을 설정할 하위 엔드포인트의 ID입니다.
(5) priority	엔드포인트의 우선순위입니다. 숫자가 낮을수록 우선됩니다.
(6) weight	엔드포인트의 가중치입니다. 숫자가 높을수록 많은 비율로 라우팅됩니다.

• loggerSystems

게이트웨이에서 사용하는 로거에 대한 설정 정보입니다. 해당 필드는 웹 어드민을 통해서만 수정이 가능합니다.

```
"loggerSystems": [{
  "path": "anylink",
  "level": "debug", (1)
  "rotatetype": "TIMEBASE", (2)
  "rotateparam": "daily", (3)
  "times": "local", (4)
  "format": "[%Y-%m-%d %L%H:%M:%S][%P][%q][%[THREAD]]: %t", (5)
  "expire": "none" (6)
}]
```

각 설정 항목에 대한 설명은 다음과 같습니다.

항목	설명
(1) level	<p>로거의 레벨입니다.</p> <ul style="list-style-type: none"> ◦ FATAL ◦ CRITICAL ◦ ERROR ◦ WARN ◦ NOTICE ◦ INFO ◦ DEBUG ◦ TRACE ◦ NONE
(2) rotatetype	<p>로거 로테이트 방식입니다.</p> <ul style="list-style-type: none"> ◦ SIZE ◦ TIMEBASE
(3) rotateparam	<p>로거 로테이트에 사용할 단위입니다.</p> <ul style="list-style-type: none"> ◦ never: 로테이트 안 함 ◦ <n>: 파일 크기가 <n> 바이트를 초과하면 로테이트 ◦ <n> K: 파일 크기가 <n> 킬로바이트를 초과하면 로테이트 ◦ <n> M: 파일 크기가 <n> 메가바이트를 초과하면 로테이트 ◦ [day,][hh:][mm]: 지정된 요일 및 시간에 로테이트 ◦ daily/weekly/monthly: 지정된 일간/주간/월간에 로테이트 ◦ <n> hours/weeks/months: 매 <n> 시간/주/월에 로테이트
(4) times	<p>로그 시간 타입입니다.</p> <ul style="list-style-type: none"> ◦ local: 현재 서버 시간 기준으로 로깅 ◦ UTC: UTC 시간 기준으로 로깅
(5) format	<p>로깅되는 포맷입니다.</p> <ul style="list-style-type: none"> ◦ [%Y-%m-%d %L%H:%M:%S]: 년-월-일 시:분:초 ◦ [%P]: 프로세스 ID ◦ [%q]: 로그 레벨 ◦ [%[THREAD]]: 스레드 정보 ◦ [%t]: 로그 내용

항목	설명
(6) expire	로그 파일의 만료기간입니다. <ul style="list-style-type: none"> ◦ none: 없음 ◦ [day,][hh:][mm] : 지정된 요일 및 시간 ◦ daily/weekly/monthly: 지정된 일간/주간/월간 ◦ <n> hours/weeks/months: 매 <n> 시간/주/월

• Monitoring

게이트웨이 모니터링에 대한 설정 정보입니다. 해당 필드는 웹 어드민을 통해서만 수정이 가능합니다.

```
"Monitoring": {
  "trace": {
    "enable" : "<bool>" (1)
    "option": {
      "fileEnable": "<bool>", (2)
      "fileName": "trace_log", (3)
      "path": "/var/log/anyapi", (4)
      "rotateParam": "30 M", (5)
      "rotateFileNum": "5", (6)
      "flushBufferSize": "32", (7)
      "interval": "3000",
      "endpoint": [ (8)
        "<grpc ip:grpc port>"
      ],
      "includeBody": "<bool>" (9)
    }
  },
  "stat": { (10)
    "enable": "<bool>",
    "option": {
      "endpoint": [
        "<grpc ip:grpc port>"
      ],
    }
  }
}
```

각 설정 항목에 대한 설명은 다음과 같습니다.

항목	설명
(1) enable	모니터링의 사용 여부입니다. <ul style="list-style-type: none"> ◦ true: 모니터링 ON ◦ false: 모니터링 OFF

항목	설명
(2) fileEnable	파일 저장 여부입니다. <ul style="list-style-type: none"> ◦ true: TRACE 정보를 파일로 저장 ◦ false: 파일 저장 사용 안 함
(3) fileName	저장될 파일 이름입니다.
(4) path	파일 저장 경로입니다.
(5) rotateParam	파일의 로테이션 크기입니다.
(6) rotateFileNum	로테이션되는 파일의 개수입니다.
(7) flushBufferSize	Span을 저장하는 버퍼 사이즈입니다.
(8) endpoint	gRPC의 IP 주소와 포트 번호입니다.
(9) includeBody	TRACE 정보에 body값의 포함 여부입니다.
(10) stat	통계 데이터의 전송 여부입니다. <ul style="list-style-type: none"> ◦ true: 전송 ◦ false: 전송 안 함

• loggerAccess

게이트웨이에서 사용하는 액세스 로거에 대한 설정 정보입니다. 해당 필드는 웹 어드민을 통해서만 수정이 가능합니다.

```
"loggerAccess": {
  "enable": "true", (1)
  "format": "%h %l %u %t %r %s %b", (2)
  "fileName": "access_log", (3)
  "path": "default", (4)
  "rotateType": "TIMEBASE", (5)
  "rotateParam": "daily", (6)
  "times": "local", (7)
  "expire": "none" (8)
}
```

각 설정 항목에 대한 설명은 다음과 같습니다.

항목	설명
(1) enable	액세스 로그의 사용 여부입니다. <ul style="list-style-type: none"> ◦ true: 사용 ◦ false: 사용 안 함

항목	설명
(2) format	<p>로깅되는 포맷입니다.</p> <ul style="list-style-type: none"> ◦ %h: 원격 호스트 주소 ◦ %l: 원격 로그인 이름 ◦ %u: Frank, HTTP 인증으로 알아낸 문서를 요청한 사용자의 ID ◦ %t: 서버 요청 처리를 마친 시간 ◦ %r: 요청의 첫 번째 줄 ◦ %s: 상태 ◦ %b: HTTP 헤더를 제외한 전송 바이트 수
(3) fileName	로그 파일의 이름입니다. (기본값: access_log)
(4) path	로그 파일이 저장되는 경로입니다. (기본값: \${GATEWAY_HOME}/protocol/HTTP/gw_id_test/logs/access_log)
(5) rotatetype	<p>로거 로테이트 방식입니다.</p> <ul style="list-style-type: none"> ◦ SIZE ◦ TIMEBASE
(6) rotateparam	<p>로거 로테이트에 사용할 단위입니다.</p> <ul style="list-style-type: none"> ◦ never: 로테이트 안 함 ◦ <n>: 파일 크기가 <n> 바이트를 초과하면 로테이트 ◦ <n> K: 파일 크기가 <n> 킬로바이트를 초과하면 로테이트 ◦ <n> M: 파일 크기가 <n> 메가바이트를 초과하면 로테이트 ◦ [day,][hh:][mm]: 지정된 요일 및 시간에 로테이트 ◦ daily/weekly/monthly: 지정된 일간/주간/월간에 로테이트 ◦ <n> hours/weeks/months: 매 <n> 시간/주/월에 로테이트
(7) times	<p>로그 시간 타입입니다.</p> <ul style="list-style-type: none"> ◦ local: 현재 서버 시간 기준으로 로깅 ◦ UTC: UTC 시간 기준으로 로깅
(8) expire	<p>로그 파일의 만료기간입니다.</p> <ul style="list-style-type: none"> ◦ none: 없음 ◦ [day,][hh:][mm] : 지정된 요일 및 시간 ◦ daily/weekly/monthly: 지정된 일간/주간/월간 ◦ <n> hours/weeks/months: 매 <n> 시간/주/월

• environment

게이트웨이의 타임아웃 관련 설정 정보입니다. 해당 필드는 웹 어드민을 통해서만 수정이 가능합니다.

```
"environment": {
  "connectionTimeout": "3000", (1)
  "readTimeout": "30000", (2)
  "failoverTimeout": "60000" (3)
}
```

각 설정 항목에 대한 설명은 다음과 같습니다.

항목	설명
(1) connectionTimeout	connectionTimeout 설정값입니다.
(2) readTimeout	readTimeout 설정값입니다.
(3) failoverTimeout	failoverTimeout 설정값입니다.

1.3.2. 디렉터리 구조

다음은 게이트웨이를 설치하고, 한번 실행한 후에 사용하는 디렉터리 구조입니다.

```
${GATEWAY_HOME}
|-- bin
|-- inc
|-- lib
|-- config
|-- protocol
|   |-- ${PROTOCOL}
|   |   |-- ${GATEWAY_NAME}
|   |-- logs
|-- uds
```

bin

AnyAPI 게이트웨이의 부팅을 위한 바이너리가 위치합니다.

inc

AnyAPI 게이트웨이 헤더 파일들이 위치합니다.

lib

AnyAPI 게이트웨이에서 사용하는 라이브러리들이 위치합니다.

config

AnyAPI 게이트웨이 설정 파일이 위치합니다.

protocol

AnyAPI 게이트웨이에서 사용하는 리소스가 위치합니다.

다음은 하위 경로에 대한 설명입니다.

하위 경로	설명
\${PROTOCOL}	게이트웨이가 사용하는 프로토콜 이름이 표시됩니다. AnyAPI에는 HTTP로 고정됩니다.
\${GATEWAY_NAME}	게이트웨이 리소스가 저장되는 디렉터리입니다.
logs	lib 디렉터리의 export 대상 디렉터리입니다.

uds

도메인 소켓 통신을 하는 경우 필요한 파일이 위치합니다.

1.3.3. 환경 변수

다음은 게이트웨이를 구동하기 위해 필요한 환경 변수에 대한 설명입니다.

환경변수	설명
GATEWAY_HOME	게이트웨이가 설치된 홈 디렉터리입니다. (필수 사항) <div>GATEWAY_HOME=/home/anylink/gateway</div>
GATEWAY_ENCODING	게이트웨이 인코딩 방식입니다. <div>GATEWAY_ENCODING=utf8</div>
GATEWAY_INACTIVE_CACHE	게이트웨이의 캐싱 여부입니다. <ul style="list-style-type: none"> ◦ true: 캐싱 비활성화 ◦ false: 캐싱 활성화 <div>GATEWAY_INACTIVE_CACHE=false</div>
LD_LIBRARY_PATH	lib 디렉터리를 export할 대상 경로입니다. <div>export LD_LIBRARY_PATH=\${LD_LIBRARY_PATH}:\${GATEWAY_HOME}/lib</div>
CURLPOT_LOW_SPEED_TIME	CURLOPT_LOW_SPEED_LIMIT 옵션으로 지정된 속도 이하로 전송될 수 있는 최대 시간입니다. 이때 초 단위로 지정하며, 해당 시간 이상 지속되면 전송이 취소됩니다.
CURLOPT_LOW_SPEED_LIMIT	최하 전송 속도입니다. 이때 초당 전송될 바이트 단위로 지정하며, 해당 속도가 CURLOPT_LOW_SPEED_TIME 옵션으로 지정된 시간만큼 지속되면 전송이 취소됩니다.

환경변수	설명
GATEWAY_BINDING_RETRYTIME	<p>바인딩 실패 후 다시 바인딩을 시도하기 전까지 대기하는 시간입니다.</p> <pre>export GATEWAY_BINDING_RETRYTIME=160000(ms)</pre>
GATEWAY_BINDING_RETRYCOUNT	<p>다시 바인딩을 시도하는 횟수입니다.</p> <p>모든 횟수를 실패하면 프로세스가 강제 종료됩니다.</p> <ul style="list-style-type: none"> ◦ 0: 재시도 없이 프로세스 종료 ◦ -1: 무한 바인딩 시도 <pre>export GATEWAY_BINDING_RETRYCOUNT=3</pre>

2. 엔드포인트 구성

본 장에서는 AnyAPI 게이트웨이의 엔드포인트 구성과 각 엔드포인트의 구성 요소들의 배포 방법에 대해 설명합니다.

2.1. 개요

엔드포인트는 실제로 외부 시스템과 연결되어 데이터를 송수신하는 기능을 담당합니다.

보통 하나의 엔드포인트는 하나의 다른 시스템과 연결할 수 있으며 하나의 게이트웨이에 엔드포인트를 여러 개 두어 여러 시스템과 연결할 수 있습니다. 엔드포인트 리소스는 엔드포인트 그룹과 엔드포인트, 두 가지의 리소스로 분리하여 관리합니다.

2.1.1. 주요 특징

엔드포인트에서는 외부에 대한 정보(IP 주소, 포트 번호, 타임아웃과 같은 운영 정보)를 관리합니다. 엔드포인트는 프로토콜마다 설정 포인트가 조금씩 다르나 AnyAPI에서는 PO와 HTTP 프로토콜만을 다룹니다.

2.1.2. AnyAPI 기본 엔드포인트

AnyAPI 게이트웨이에서는 기본적으로 4개의 엔드포인트가 사전에 staticResources 내에 정의되어 있으며 웹 어드민을 통해 서버를 추가할 때마다 dynamicResources에 엔드포인트들이 추가됩니다.

다음은 AnyAPI 게이트웨이에서 기본적으로 설정되어 있는 4가지 엔드포인트에 대한 설명입니다.

- **poClientEndpoint**

poClientEndpoint는 내부 관리를 위해서 Master와 통신하기 위해 만들어진 클라이언트 엔드포인트입니다. 게이트웨이는 부팅할 때 해당 엔드포인트를 통하여 Admin과 연결을 맺고 GatewayPost 서비스를 요청하여 연결을 수립합니다. 연결이 수립되면 Admin은 들고 있는 전체 설정 정보를 배포하게 되며 이후 설정이 변경될 때마다 부분 배포가 이루어집니다. 또한 실제 API가 동작하게 될 때 이 연결을 통해서 Master에 인증/인가 등을 요청합니다.

- **poServerEndpoint**

poServerEndpoint는 Master로부터 라우팅 정보를 배포받기 위해 사용하는 서버 엔드포인트입니다. 내부 관리 통신을 위한 포트를 설정하여 설정 파일을 배포하거나 받을 수 있습니다.

- **httpEndpoint / httpsEndpoint**

httpEndpoint와 httpsEndpoint는 API 호출을 위한 클라이언트 외부 요청을 처리하는 주 포트에 대한 설정입니다. httpEndpoint는 HTTP, httpsEndpoint는 HTTPS에 대한 포트입니다.

2.2. 엔드포인트 설정

엔드포인트는 게이트웨이의 설정 파일 내에 복수의 엔드포인트를 설정할 수 있습니다. (현재 JSON 형식만 지원)

다음은 엔드포인트의 설정 예시 및 설정 항목에 대한 설명입니다.

• PO 엔드포인트 설정 예시

```
"endpoint": {
  "physicalName": "poClientEndpoint",
  "protocol": "HTTP",
  "direction": "Outbound",
  "bootState": "Running",
  "connectType": "Client",
  "httpUrl": {
    "split": true,
    "scheme": "http",
    "host": "${MS_IP}", (1)
    "port": "${MS_PORT}" (2)
  }
}
```

항목	설명
(1) host	연결할 외부 주소입니다. 단, 클라이언트일 때 설정합니다. (보통 JEUS의 Master Server IP를 사용)
(2) port	연결에 사용할 포트 번호입니다. (보통 JEUS의 MS HTTP 포트를 사용) 게이트웨이의 서버, 클라이언트 모드에 따라 의미가 다릅니다. <ul style="list-style-type: none">◦ SERVER: 연결할 외부 포트◦ CLIENT: 게이트웨이에서 열 내부 포트

• HTTPS 엔드포인트 설정 예시

```
"endpoint": {
  "physicalName": "httpsEndpoint",
  "protocol": "HTTP",
  "direction": "Inbound",
  "bootState": "Running",
  "connectType": "Server",
  "connection_pool": {
    "max": "-1" // 엔드포인트 최대 연결 수
  },
  "idleTimeout": "360000", (1)
  "useSsl": true, (2)
  "ssl": { (3)
    "keystore": { (4)
      "storeType": "PEM", (5)
      "storeLocation": "${SSL_STORE}" (6)
    }
  },
}
```

```

    "httpUrl": {
      "scheme": "https",
      "port": "${HTTPS_PORT}" (7)
    },
    "rulesetId": "ApiApplication.ApiServiceGroup.ApiDefaultRuleChain",
    "errorRulesetId": "anylink.system.ApiGatewayErrorRuleSet"
  }

```

항목	설명
(1) idleTimeout	소켓에 응답이 없을 경우 세션이 종료되는 시간입니다. (단위: ms, 기본값: 360000)
(2) useSsl	엔드포인트의 SSL 적용 여부입니다.
(3) ssl	엔드포인트에서 사용할 SSL에 대한 세부 정보입니다.
(4) keystore	SSL 관련 Key Store 정보입니다. 해당 필드는 SSL 인증에 사용될 Key Store 파일 설정과 연관됩니다.
(5) storeType	사용할 인증서 타입입니다. (현재 PEM 타입만 가능)
(6) storeLocation	인증서가 저장된 파일 경로입니다.
(7) port	내부 HTTPS 포트 번호입니다.

• 공통 엔드포인트 설정 (PO, HTTP)

항목	설명
physicalName	엔드포인트의 물리적인 이름입니다.
protocol	엔드포인트에서 사용하는 프로토콜입니다.
direction	엔드포인트의 방향입니다. <ul style="list-style-type: none"> ◦ OUTBOUND: 외부로 나가는 엔드포인트 ◦ INBOUND: 외부에서 들어오는 엔드포인트 ◦ BOTH: 외부와 양방향으로 통신이 가능한 엔드포인트
bootState	게이트웨이 부팅 시 엔드포인트의 초기 상태입니다. <ul style="list-style-type: none"> ◦ Running: 부팅할 때 자동으로 시작합니다. ◦ Stopped: 부팅할 때 시작하지 않습니다.
connectType	게이트웨이의 연결 타입입니다. <ul style="list-style-type: none"> ◦ CLIENT: 클라이언트 타입의 엔드포인트 ◦ SERVER: 서버 타입의 엔드포인트

항목	설명
connection_pool	<p>게이트웨이의 연결 풀입니다. 연결 풀은 엔드포인트가 여러 개의 물리적인 통신 연결을 가질 수 있기 때문에 이러한 물리적인 연결을 어떻게 관리할지 설정할 수 있습니다.</p> <ul style="list-style-type: none"> ◦ min: 연결 풀이 최소로 들고 있어야 하는 연결의 개수를 양의 정수로 지정합니다. 항상 이 연결 수 만큼 유지해야하므로 부팅이 된 엔드포인트는 이 값의 연결이 생길 때까지 연결을 시도합니다. 단, 클라이언트 타임아웃 때만 동작합니다. ◦ max: 연결 풀이 최대로 유지할 수 있는 연결 개수를 양의 정수로 지정합니다.
rulesetId	해당 엔드포인트로 요청이 들어왔을 때 실행시킬 룰입니다.
errorRulesetId	해당 엔드포인트로 요청을 처리하는 도중 에러가 발생하였을 때 처리할 룰입니다.

2.2.1. SSL Passthrough 설정

SSL Passthrough는 수신한 데이터를 가공하지 않고 그대로 다른 시스템에 전달하여, 해당 시스템에서 직접 SSL 처리를 하도록 하는 기능입니다.

기존에는 SSL 요청을 게이트웨이에 저장된 인증서를 이용해 읽은 후 다른 시스템에 요청하는 방식이었으나, SSL Passthrough 기능을 사용하면 인증서를 관리하지 않고도 다른 시스템과 SSL 통신을 할 수 있습니다.

단, SSL Passthrough 기능을 사용할 경우 게이트웨이는 트래픽을 단순히 포워딩만 하기 때문에 트래픽 분석, 모니터링, 로깅, 인증서 관리가 불가능합니다. 따라서 외부 시스템에서 사용할 SSL 인증서를 별도로 준비해야 하며, 설정을 통해 새로운 엔드포인트를 추가하고 포트를 개방하여 직접 외부 시스템을 연결해야 합니다.

다음은 SSL Passthrough를 사용하기 위한 엔드포인트 설정 예시입니다.

```
"endpoint": {
  "physicalName": "sslPassthroughEndpoint",
  "direction": "Inbound",
  "bootState": "Running",
  "connectType": "Server",
  "connection_pool": {
    "max": "-1" // 엔드포인트 최대 연결 수
  },
  "httpUrl": {
    "scheme": "https",
    "port": "${LISTEN_PORT}" // 게이트웨이가 열 포트 번호
  },
  "ssl_mode" : "PASSTHROUGH", (1)
  "stream_rule_chains": { (2)
    "rule": [{
      "name": "SSLPassthroughRule",
      "config": {
        "target": "${TARGET_ADDRESS}" // 외부 시스템 시스템 주소
      }
    }]
  }
}
```

항목	설명
(1) ssl_mode	<p>SSL 처리 방식입니다. (기본값: TERMINATE)</p> <p>SSL Passthrough 기능을 사용하기 위해서는 반드시 "PASSTHROUGH"로 설정해야 합니다.</p>
(2) stream_rule_chains	<p>요청이 들어왔을 때 실행할 룰체인 설정입니다.</p> <ul style="list-style-type: none"> ◦ rule.name: 실행할 룰 이름입니다. 반드시 "SSLPassthroughRule"로 설정해야 합니다. ◦ rule.config.target: 연결할 외부 시스템의 주소입니다. (예: 1.2.3.4:80)

3. 모니터링

본 장에서는 AnyAPI 게이트웨이의 모니터링에 대한 전반적인 소개와 설정 파일에 대해 설명합니다.

3.1. 모니터링 기본 개념

3.1.1. 메트릭(Metric)

AnyAPI 엔진에서 발생하는 정보로 AnyAPI 운영에 있어 중요한 지표를 통계적으로 가공한 데이터입니다

다음은 메트릭의 종류별 개념과 AnyAPI 게이트웨이에서 수집하는 항목에 대한 설명입니다.

• 카운터(Counter)

단조롭게 증가하는 메트릭입니다. 주로 이벤트 발생 횟수나 특정 작업의 횟수를 측정합니다. (예: 요청 처리 횟수, 오류 횟수)

메트릭 이름	설명
http.server.gateway_total_time	게이트웨이 내부 로직 처리 시간 측정
http.server.total_requests	리소스 요청 횟수 측정
http.server.fail_over	라우팅 실패 횟수 측정
http.server.no_queueing_delay	작업 큐 대기시간을 제외한 게이트웨이 내부 로직 처리 시간 측정
http.server.response_count	응답 코드별 횟수 측정
http.server.success_rate	100~300 범위의 응답 코드가 발생한 횟수 측정
http.server.time_out	라우팅에서 서비스의 read timeout 발생 횟수 측정
http.server.response_latency	응답 코드별 지연 시간(latency) 측정

• 게이지(Gauge)

특정 시점의 값을 측정하며, 값이 증가하거나 감소할 수 있습니다. (예: 메모리 사용량, CPU 사용률, 디스크 공간)

메트릭 이름	설명
server.thread_pool.current_queue_size	현재 스레드 풀의 큐 크기
server.thread_pool.current_thread_size	현재 스레드 풀의 스레드 수
server.thread_pool.current_worker_size	현재 작업 중인 스레드 수
server.thread_pool.max_queue_size	스레드 풀의 최대 큐 크기
server.thread_pool.max_thread_size	스레드 풀의 최대 스레드 수

• 히스토그램(Histogram)

데이터가 특정 구간에 어떻게 분포하는지 나타냅니다. (예: 요청 응답 시간, 파일 크기 분포)

메트릭 이름	설명
http.server.duration	리소스별 지연 시간(latency) 측정

- **타이머(Timer)**

시간 측정용 메트릭입니다. 주로 요청 처리 시간이나 작업 실행 시간을 측정합니다. 히스토그램과 유사하지만, 이벤트 지속 시간을 다룹니다.

3.1.2. 트레이스(Trace)

AnyAPI에서 발생한 요청 또는 거래의 흐름을 나타내는 데이터입니다. 이 흐름을 추적하기 위해 다양한 메타 데이터 (예: 타임스탬프, 서비스 이름, 상태 코드 등)가 포함되어 있습니다.

다음은 트레이스 데이터의 구성 요소에 대한 설명입니다.

- **스팬(Span)**

트레이스는 여러 개의 스패스로 구성되며, 각 스패스는 시스템 내 작업이나 서비스를 나타냅니다. 스패스는 시작/종료 시간, 상태, 태그, 로그 등의 정보를 포함하고, 트랜잭션 흐름에 따라 연결됩니다.

- **트레이스 ID**

모든 트레이스에는 고유한 트레이스 ID가 할당됩니다. 이를 통해 동일한 트랜잭션에 관련된 여러 스패스를 하나의 트레이스로 묶을 수 있습니다.

- **메타 데이터**

각 스패스는 메타 데이터를 포함할 수 있습니다. 예를 들어 서비스 이름, API 엔드포인트, 응답 시간, 오류 상태 등을 기록하여 성능 분석과 문제 진단에 활용됩니다.

3.1.3. gRPC

구글이 개발한 오픈 소스 원격 프로시저 호출(RPC) 시스템입니다. gRPC는 HTTP/2를 기반으로 하며, 프로토콜 버퍼를 사용해 메시지를 직렬화하여 효율적인 통신을 지원합니다.

3.1.4. OpenTelemetry

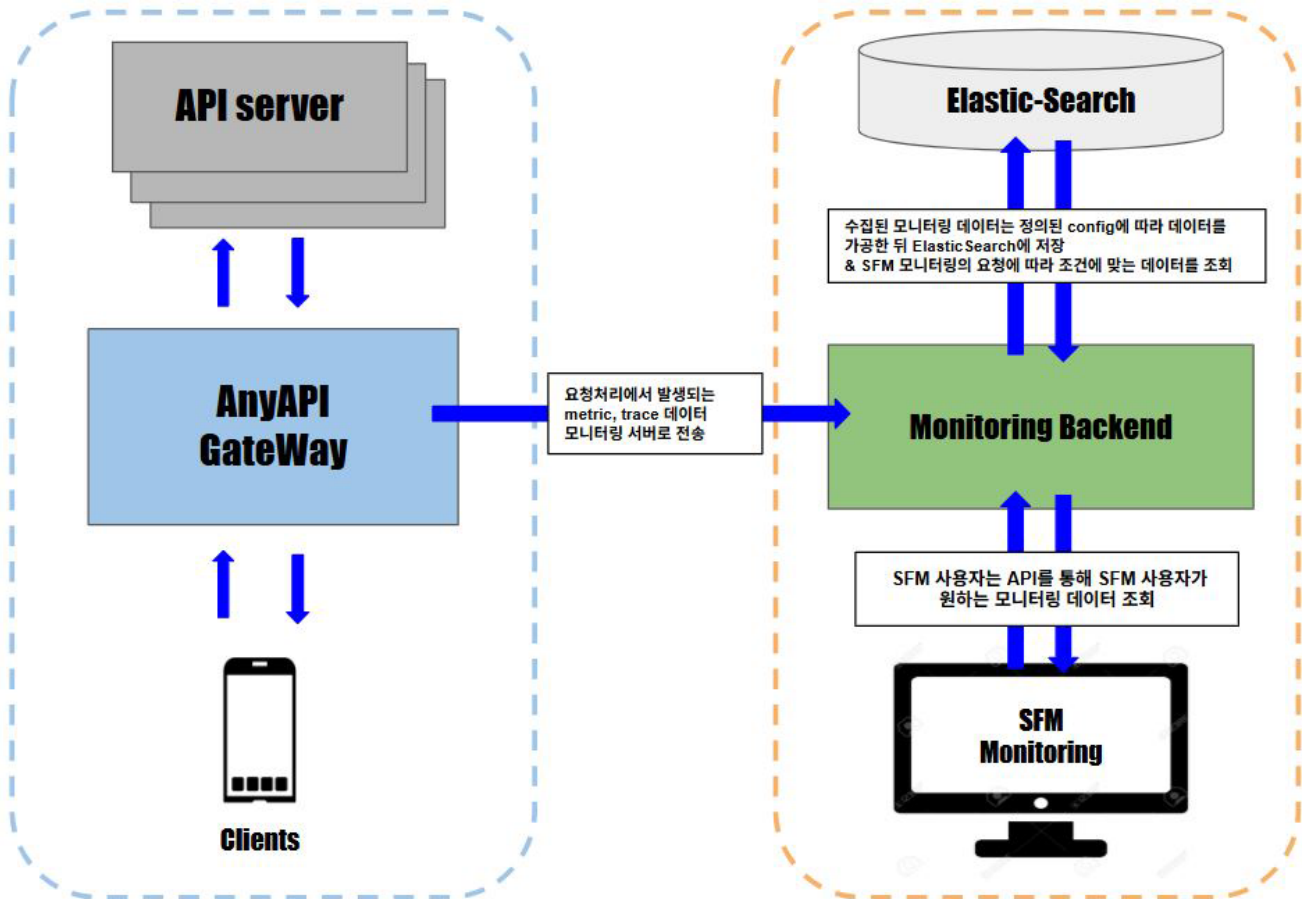
분산 시스템 환경에서 애플리케이션 성능을 모니터링하고 추적하기 위한 오픈 소스입니다.

OpenTelemetry는 추적(Tracing), 메트릭(Metrics), 로그(Logs)를 위한 통합 표준을 제공하며, openTelemetry-proto를 통해 protobuf 기반의 표준 정의를 확인할 수 있습니다.

3.2. 모니터링 구조 및 흐름

3.2.1. 모니터링 전체 구조

AnyAPI의 모니터링은 데이터를 수집하는 영역과 사용자가 데이터를 조회하는 영역으로 구성됩니다.



• 모니터링 데이터 수집

AnyAPI 엔진은 gRPC를 통해 주기적으로 메트릭과 트레이스 데이터를 모니터링 백엔드로 전송합니다.

이때 메트릭 데이터는 설정된 구성(config)에 따라 가공하여 Elastic-Search에 저장하고, 트레이스 데이터는 트레이스 ID와 스패 ID를 기반으로 거래 흐름을 추적할 수 있도록 구조화하여 Elastic-Search에 저장합니다.

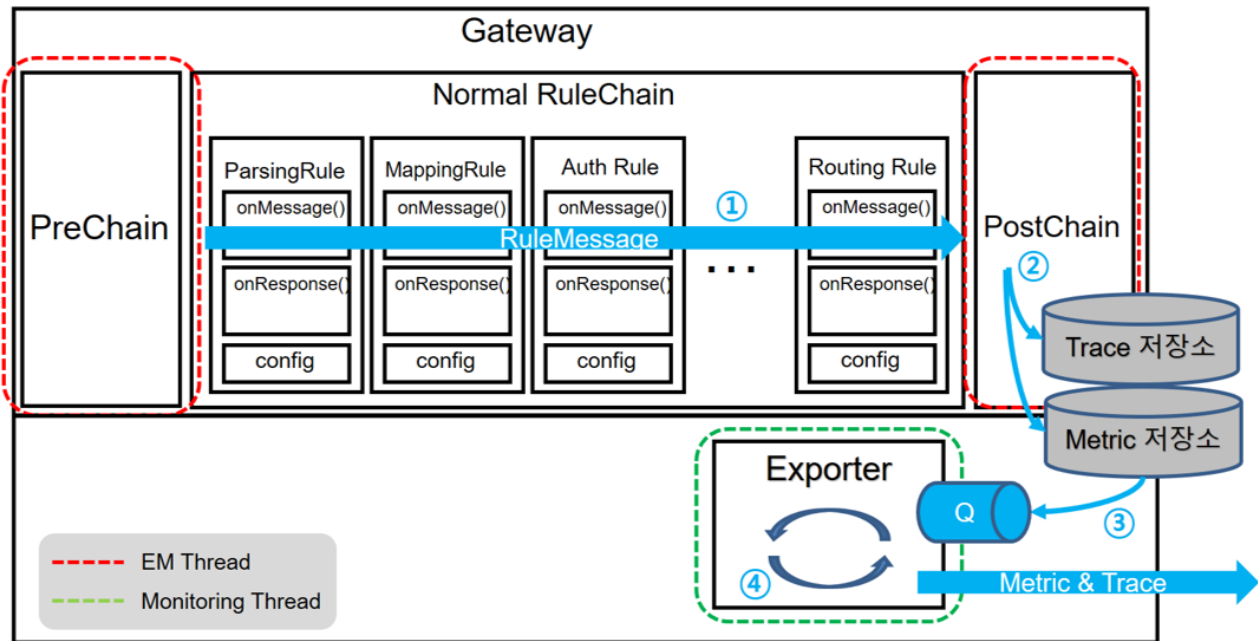
• 모니터링 데이터 조회

SFM 모니터링은 백엔드로부터 수집된 메트릭 데이터를 주기적으로 불러와 AnyAPI 게이트웨이 상태를 실시간으로 확인합니다. 이때 필터링을 통해 원하는 메트릭 및 트레이스 데이터를 조회할 수 있습니다.

3.2.2. 모니터링 데이터 수집 과정

위 그림에서는 AnyAPI GW에서 metric과 trace 데이터가 어떻게 생성되어 모니터링 백엔드에게 전송되는지를 표현한 그림이다. GW에서 어떻게 trace와 metric 데이터를 생성하여 모니터링 서버로 데이터를 전송하는 과정(1 → 2 → 3 → 4)에 대해 자세히 설명하겠습니다.

AnyAPI 게이트웨이에서 메트릭과 트레이스 데이터가 생성되어 모니터링 백엔드로 전송됩니다.



각 단계별 주요 작업은 다음과 같습니다.

1. 게이트웨이는 요청을 처리할 때 preChain → RuleChain(ParsingRule, MappingRule 포함) → postChain 순으로 진행합니다.

이 과정에서 RuleMessage 객체는 요청 처리에 필요한 컨텍스트 정보를 유지하며, 요청이 완료될 때까지 이를 보관합니다. 또한 내부에 TraceMessage와 MetricMessage 객체를 포함하고 있어 요청 처리 중 발생하는 트레이스 및 메트릭 데이터를 갱신하거나 추가합니다.

```
RuleMessage
├── TraceMessage # Rule 처리 과정 중 생성되는 트레이스 데이터
└── MetricMessage # Rule 처리 과정 중 생성되는 메트릭 데이터
```

2. postChain에서 요청이 EM 스레드에 의해 처리되면, RuleMessage에 포함된 메트릭과 트레이스 데이터는 각각 별도의 저장소에 저장됩니다.

- 메트릭 저장소

메트릭 데이터는 HashMap<String, Metric> 구조에 저장되며, 리소스별 호출 횟수와 지연 시간(latency) 등을 기록합니다.

```
HashMap<리소스-이름, metric>
├── 리소스 A, {호출 횟수, 지연 시간, ...}
├── 리소스 B, {호출 횟수, 지연 시간, ...}
└── 리소스 C, {호출 횟수, 지연 시간, ...}
```

- 트레이스 저장소

트레이스 데이터는 하나의 트레이스 ID를 기준으로 여러 개의 스팸으로 구성되며 Vector 구조에 저장됩니다.

```
Vector<Span>
└── | span1{trace-id:A} | span2{trace-id:A} | span1{trace-id:B} | ....
```

3. 수집된 메트릭 및 트레이스 데이터는 일정 주기로 Exporter에게 전달됩니다.

Exporter는 opentelemetry-proto 기반으로 구현된 gRPC 서비스 객체로, 수집된 데이터를 모니터링 서버로 전송하는 역할을 합니다. 이 데이터는 EM 스레드에 의해 Exporter로 전달되며, 해당 과정은 ThreadPool과 유사한 방식으로 처리됩니다.

EM 스레드는 일정 주기마다 모니터링 전용 스레드의 작업 큐에 데이터 전송 작업을 등록합니다. 단, 작업 큐가 가득 차 있는 경우 해당 작업은 큐에 등록되지 않고 삭제되며, 해당 주기의 메트릭 및 트레이스 전송은 생략됩니다.

4. 모니터링 전용 스레드는 작업 큐에서 하나의 작업(Work)을 꺼내어 처리합니다.

이때 메트릭 및 트레이스 데이터는 OpenTelemetry-protobuf 형식에 맞게 객체화되며, gRPC 서비스를 통해 직렬화된 형태로 모니터링 서버에 전송합니다.

3.2.3. 모니터링 데이터 전달 과정


게이트웨이에서 발생한 메트릭 및 트레이스 데이터는 opentelemetry-proto에서 정의한 서비스 형식에 따라 직렬화되어 모니터링 서버로 전송됩니다.



3.2.4. 모니터링 서버 처리 과정

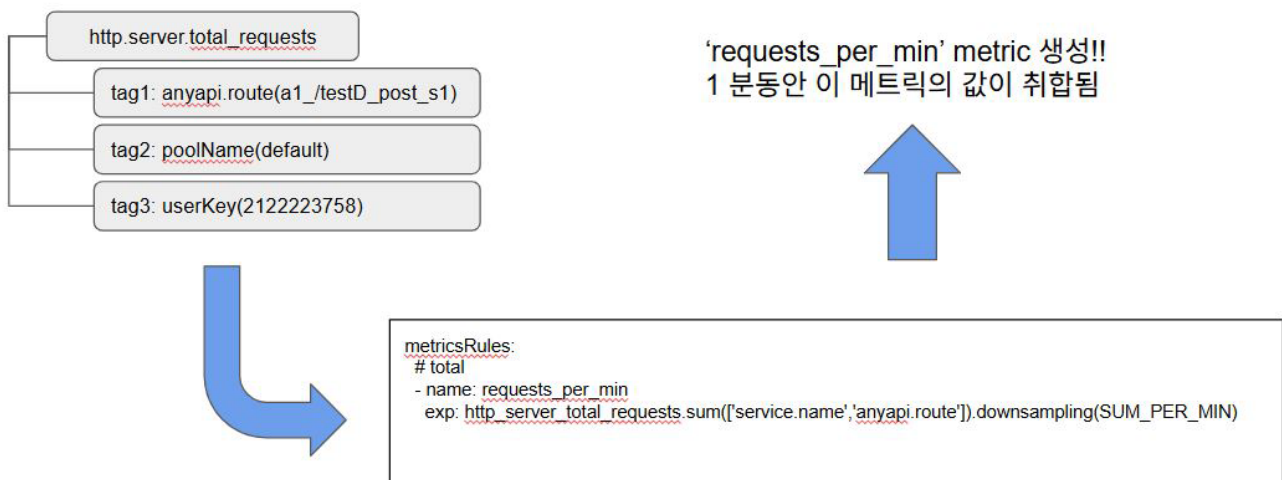
게이트웨이에서 전송된 데이터는 모니터링 서버에서 내부적으로 가공되어 저장소에 저장됩니다.

"\${모니터링 서버 설치 디렉터리}/config/otel-oc-rules" 디렉터리에는 AnyAPI 게이트웨이의 메트릭 데이터를 처리하는 방식이 설정되어 있으며, 이 구성은 key/value 형식으로 정의됩니다. key에는 모니터링 서버에서 사용하는 메트릭 이름이 지정되며, value에는 해당 메트릭을 처리하기 위한 함수 형식의 변수가 설정됩니다.



```
filter: "{ tags -> tags['provider.type'] == 'anyapi' }" # The OpenTelemetry job name
expSuffix: endpoint( ['service.name'], ['anyapi.route'], Layer.GENERAL)
metricPrefix: anyapi_endpoint
metricsRules:
  # total
  - name: success_rate
    exp: "http_server_success_rate.sum(['service.name','anyapi.route','type']).tag((tags -> if (tags['type'] == 'sum') {tags
  - name: requests
    exp: http_server_total_requests.sum(['service.name','anyapi.route']).downsampling(SUM)
  - name: requests_method
    exp: http_server_total_requests.sum(['service.name','anyapi.route','http.method']).downsampling(SUM)
  - name: requests_per_min
    exp: http_server_total_requests.sum(['service.name','anyapi.route']).downsampling(SUM_PER_MIN)
  - name: responses_status_code
    exp: http_server_duration_count.sum(['service.name','anyapi.route','http.status_code']).downsampling(SUM)
```

게이트웨이에서 수신된 데이터는 메타 데이터를 기준으로 분류되고, 이를 기반으로 모니터링 서버 내부 포맷에 맞는 메트릭 데이터로 재정의되어 저장됩니다. 이후 이 데이터를 활용해 모니터링 지표를 조회하거나 분석할 수 있습니다.



3.3. 게이트웨이 설정

3.3.1. 트레이스 설정 예시

게이트웨이에서 트레이스 데이터를 수집하고 전송하기 위한 설정 예시는 다음과 같습니다.

```

"monitoring": {
  "trace": {
    "enable": "false",
    "option": {
      "fileEnable": "false",
      "fileName": "${HOSTNAME}_trace_log",
      "path": "/home/gateway/gateway21/trace/${NODENAME}",
      "rotateParam": "10 M",
      "rotateFileNum": "10",
      "interval": "3000",
      "flush_buffer_size": "32",
      "endpoint": [
        "192.168.15.185:11800"
      ],
      "includeBody": "false",
      "encryptEnable": "true",
      "filter": {
        "response4XX": "true",
        "response5XX": "true",
        "latencyOver": "300",
        "IncludePath": "/abc", "/hello",
      }
    }
  }
}

```

항목	타입	설명
trace.enable	string	트레이스 데이터를 생성할지 여부입니다. <ul style="list-style-type: none"> ◦ true: 생성함 ◦ false: 생성하지 않음
trace.option.interval	string	스팬을 gRPC로 송출하는 주기입니다. (단위: ms)
trace.option.flush_buffer_size	string	스팬을 보관하는 버퍼의 크기입니다.
trace.option.endpoint	list<string>	gRPC로 송출할 모니터링 서버의 엔드포인트 리스트입니다. (failover는 지원하지 않음)
trace.option.includeBody	string	HTTP 본문(body) 데이터를 트레이스 속성에 포함할지 여부입니다. <ul style="list-style-type: none"> ◦ true: 포함함 ◦ false: 포함하지 않음
trace.option.encryptEnable	string	본문(body) 데이터를 암호화하여 저장할지 여부입니다. <ul style="list-style-type: none"> ◦ true: 암호화함 ◦ false: 암호화하지 않음

항목	타입	설명
trace.option.filter.response4XX	string	HTTP 4XX 응답 코드를 트레이스에 포함할지 여부입니다. ◦ true: 포함함 ◦ false: 포함하지 않음
trace.option.filter.response5XX	string	HTTP 5XX 응답 코드를 트레이스에 포함할지 여부입니다. ◦ true: 포함함 ◦ false: 포함하지 않음
trace.option.filter.latencyOver	string	트레이스에 포함할 요청의 최소 지연 시간입니다.
trace.option.filter.IncludePath	vector<string>	트레이스 데이터 수집 대상이 되는 요청 경로입니다.
trace.option.fileEnable	string	트레이스 로그 데이터를 파일로 저장할지 여부입니다. ◦ true: 파일로 저장함 ◦ false: 파일로 저장하지 않음
trace.option.fileName	string	트레이스 로그 파일의 이름입니다.
trace.option.path	string	트레이스 로그 파일이 저장되는 디렉터리 경로입니다.
trace.option.rotateParam	string	단일 트레이스 로그 파일의 최대 크기입니다.
trace.option.rotateFileNum	string	보관할 트레이스 로그 파일의 최대 개수입니다.

3.3.2. 메트릭 설정 예시

게이트웨이에서 메트릭 데이터를 수집하고 전송하기 위한 설정 예시는 다음과 같습니다.

```

"monitoring": {
  "stat": {
    "enable": "true",
    "option": {
      "endpoint": ["192.168.15.130:4317"],
      "histogram_explicit_bounds": ["0", "10000"],
      "interval": "4000"
    }
  }
}

```

항목	타입	설명
stat.enable	string	메트릭 데이터를 생성할지 여부입니다. ◦ true: 생성함 ◦ false: 생성하지 않음

항목	타입	설명
stat.option.endpoint	list<string>	gRPC로 송출할 모니터링 서버의 엔드포인트 리스트입니다.
stat.option.histogram_explicit_bounds	list<string>	Histogram 메트릭의 구간 경계값 리스트입니다. (x축 구간 설정)
stat.option.interval	string	메트릭 데이터를 송출하는 주기입니다.

3.4. 모니터링 Kafka 연동

기존 AnyAPI 게이트웨이는 모니터링 데이터를 직접 모니터링 서버로 전송합니다. 이 방식은 실시간 데이터 확인이 가능하나, 네트워크 또는 서버 장애 시 데이터 유실 위험이 있습니다.

이를 보완하기 위해 Kafka를 활용한 안정적인 데이터 전달 구조를 도입하여, 데이터 손실 없이 안정적인 처리를 지원합니다.

3.4.1. 구성 및 동작 방식

Kafka 연동 방식에서는 기존 모니터링 서버 외에도 메시지 브로커 역할의 Kafka 브로커, AnyAPI 게이트웨이와 Kafka 간의 프로토콜 변환을 담당하는 OpenTelemetry Collector가 추가로 필요합니다.

• Kafka

분산 스트리밍 플랫폼으로 대규모 데이터를 실시간으로 수집, 저장, 처리합니다. 이때 고성능, 확장성, 내결함성을 제공하여 데이터를 빠르고 안정적으로 전송합니다.

Kafka는 Producer가 특정 토픽으로 데이터를 보내고 Consumer가 이를 수신하는 구조로 동작합니다. 여러 브로커로 구성된 클러스터가 데이터를 분산 저장하며, 각 토픽은 여러 파티션으로 나누어져 메시지 순서를 보장합니다. 또한 데이터를 여러 서버에 복제하여 시스템 장애 시에도 손실을 방지합니다.

• OpenTelemetry Collector

분산된 서비스에서 수집된 텔레메트리 데이터를 중앙에서 처리하고 전달합니다.

주요 구성 요소는 다음과 같습니다.

구성 요소	설명
Receiver	데이터 소스를 정의하며 OTLP, Jaeger, Prometheus, Zipkin 등 다양한 포맷을 지원합니다.
Processor	수집된 데이터를 필터링, 샘플링, 배치 처리, 변환합니다. 예를 들어 특정 로그 제외나 메트릭 집계 가능합니다.
Exporter	처리된 데이터를 Jaeger, Prometheus, Elasticsearch, AWS X-Ray 등 원하는 백엔드 시스템으로 전송합니다.
Extension	인증, 로깅, 모니터링 등 추가 기능을 제공합니다.

다음은 Kafka 연동 시 모니터링 데이터가 처리되고 전달되는 과정입니다.

```
AnyAPI 게이트웨이
  ↓ (OTLP)
OTEL Collector (Producer 모드)
  ↓ (Kafka 프로토콜)
Kafka
  ↓ (Kafka 프로토콜)
OTEL Collector (Consumer 모드)
  ↓ (OTLP Exporter)
모니터링 서버
```

1. AnyAPI 게이트웨이는 메트릭, 트레이스 등 모니터링 데이터를 OTLP(OpenTelemetry Protocol) 형식으로 OTEL Collector에 전송합니다.
2. OTEL Collector(Producer 모드)는 수신한 데이터를 Kafka 프로토콜로 변환하여 Kafka 브로커에 전송합니다. 이 과정에서 필터링, 변환, 압축 작업을 수행할 수 있습니다.
3. Kafka는 데이터를 안정적으로 저장하고, 대기 중인 OTEL Collector로 전달합니다. 높은 내결함성과 데이터 처리량으로 데이터 유실을 방지합니다.
4. OTEL Collector(Consumer 모드)는 Kafka에서 데이터를 받아 OTLP 형식으로 다시 변환한 후, OTLP Exporter를 통해 최종 모니터링 서버로 전송합니다.
5. 모니터링 서버는 수집된 데이터를 실시간으로 처리하고, 대시보드 및 알림 시스템을 통해 운영자에게 시각화된 정보를 제공합니다.

3.4.2. 설정 방법

모니터링 Kafka 연동을 위해 Kafka와 OpenTelemetry가 베어메탈 환경이나 도커 등 가상화 플랫폼에 설치되어 있어야 합니다. 또한 연동 전에 메트릭용 "sfm-anyapi-metric" 토픽과 트레이스용 "sfm-anyapi-trace" 토픽을 Kafka에 사전 등록해야 합니다.

OpenTelemetry Collector 설정 파일은 트레이스와 메트릭 데이터를 수집 및 처리한 뒤 Kafka로 전송하거나, Kafka에서 받아 최종 목적지로 전달하는 방식을 정의합니다.

3.4.2.1. otel-collector-producer-config.yaml

다음은 OpenTelemetry Collector(Producer 모드)가 데이터를 수집해 Kafka로 전송하는 설정 예시입니다.

```
receivers:
  otlp:
    protocols:
      grpc: anyapi_gw:4317

exporters:
  kafka/traces:
    brokers: [kafka:9092]
    topic: "sfm-anyapi-trace"
    encoding: "otlp_proto"
```

```

kafka/metrics:
  brokers: [kafka:9092]
  topic: "sfm-anyapi-metric"
  encoding: "otlp_proto"

processors:
  batch:

service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [batch]
      exporters: [kafka/traces]
    metrics:
      receivers: [otlp]
      processors: [batch]
      exporters: [kafka/metrics]

```

• receivers

OpenTelemetry Collector가 데이터를 수신하는 방식을 정의하며, AnyAPI 게이트웨이에서 전송하는 모니터링 데이터를 받는 역할을 합니다.

항목	설명
otlp.protocols.grpc	gRPC 프로토콜을 통해 메트릭 및 트레이스 데이터를 수신할 주소입니다.

• exporters

OpenTelemetry Collector가 receivers에서 받은 데이터를 재가공하여 지정된 엔드포인트로 전송하는 역할을 합니다.

항목	설명
kafka/traces.brokers	트레이스 데이터를 전송할 Kafka 브로커 주소입니다. 예시에서는 "kafka:9092"로 지정하여 Kafka 클러스터와 연결합니다.
kafka/traces.topic	트레이스 데이터를 전송할 Kafka 토픽 이름입니다. 예시에서는 "sfm-anyapi-trace"라는 토픽을 사용합니다.
kafka/traces.encoding	트레이스 데이터를 Kafka로 전송할 때 사용할 인코딩 방식입니다. 예시에서는 OTLP 프로토콜 형식(otlp_proto)을 사용합니다.
kafka/metrics.brokers	메트릭 데이터를 전송할 Kafka 브로커 주소입니다. 예시에서는 "kafka:9092"로 지정하여 Kafka 클러스터와 연결합니다.

항목	설명
kafka/metrics.topic	메트릭 데이터를 전송할 Kafka 토픽 이름입니다. 예시에서는 "sfm-anyapi-metric"라는 토픽을 사용합니다.
kafka/metrics.encoding	메트릭 데이터를 데이터를 Kafka로 전송할 때 사용할 인코딩 방식입니다. 예시에서는 OTLP 프로토콜 형식(otlp_proto)을 사용합니다.

• processors

OpenTelemetry Collector의 receivers에서 수신한 데이터를 exporters로 전송할 때 데이터를 전처리하거나 변형하는 역할을 합니다.

항목	설명
batch	수신한 데이터를 일정 시간 또는 일정 개수 단위로 묶어 처리하기 위한 설정 항목입니다. 데이터를 일괄 처리하여 전송 성능을 최적화하며, 네트워크 지연을 줄이고 대량 데이터를 효율적으로 처리할 수 있도록 합니다.

• service

OpenTelemetry Collector의 전체 동작을 정의하는 최상위 구성 요소로 Collector가 실행할 파이프라인, 활성화할 확장 기능, 전체 제어 설정을 포함합니다.

항목	설명
pipelines.traces.receivers	트레이스 데이터를 수신하는 receivers 항목입니다.
pipelines.traces.processors	트레이스 데이터에 적용할 processors 항목입니다.
pipelines.traces.exporters	트레이스 데이터를 전송할 exporters 항목입니다.
pipelines.metrics.receivers	메트릭 데이터를 수신하는 receivers 항목입니다.
pipelines.metrics.processors	메트릭 데이터에 적용할 processors 항목입니다.
pipelines.metrics.exporters	메트릭 데이터를 전송할 exporters 항목입니다.

3.4.2.2. otel-collector-consumer.yaml

다음은 OpenTelemetry Collector(Consumer 모드)에서 Kafka로부터 데이터를 받아 모니터링 서버로 전달하는 설정 예시입니다.

```
receivers:
  kafka/traces:
    brokers: [kafka:9092]
    topic: "sfm-anyapi-trace"
    encoding: "otlp_proto"
```

```

kafka/metrics:
  brokers: [kafka:9092]
  topic: "sfm-anyapi-metric"
  encoding: "otlp_proto"

processors:
  batch:

exporters:
  otlp:
    endpoint: oap:11800
    tls:
      insecure: false
  debug:
    verbosity: detailed

service:
  pipelines:
    traces:
      receivers: [kafka/traces]
      processors: [batch]
      exporters: [debug, otlp]
    metrics:
      receivers: [kafka/metrics]
      processors: [batch]
      exporters: [debug, otlp]

```

• receivers

Kafka 브로커가 전송한 데이터를 수신하는 역할을 합니다.

항목	설명
kafka/traces.brokers	트레이스 데이터를 수신할 Kafka 브로커의 엔드포인트입니다.
kafka/traces.topic	트레이스 데이터를 수신할 Kafka 토픽입니다.
kafka/traces.encoding	트레이스 데이터를 수신할 때 사용할 인코딩 방식입니다.
kafka/metrics.brokers	메트릭 데이터를 수신할 Kafka 브로커의 엔드포인트입니다.
kafka/metrics.topic	메트릭 데이터를 수신할 Kafka 토픽입니다.
kafka/metrics.encoding	메트릭 데이터를 수신할 때 사용할 인코딩 방식입니다.

• processors

OpenTelemetry Collector의 receivers에서 수신한 데이터를 exporters로 전송할 때 데이터를 전처리하거나 변형하는 역할을 합니다.

항목	설명
batch	수신한 데이터를 일정 시간 또는 일정 개수 단위로 묶어 처리하기 위한 설정 항목입니다. 데이터를 일괄 처리하여 전송 성능을 최적화하며, 네트워크 지연을 줄이고 대량 데이터를 효율적으로 처리할 수 있도록 합니다.

- **exporters**

Kafka 브로커로부터 수신한 데이터를 SFM 모니터링(OAP 서버)으로 전송하는 역할을 합니다.

항목	설명
otlp.endpoint	데이터를 전송할 외부 시스템의 주소입니다.
otlp.tls	데이터 전송 시 TLS의 적용 여부입니다.
debug	데이터 전송 과정에서 출력되는 로그의 상세 수준입니다.

- **service**

OpenTelemetry Collector의 전체 동작을 정의하는 최상위 구성 요소로 Collector가 실행할 파이프라인, 활성화할 확장 기능, 전체 제어 설정을 포함합니다.

항목	설명
pipelines.traces.receivers	트레이스 데이터를 수신하는 receivers 항목입니다.
pipelines.traces.processors	트레이스 데이터에 적용할 processors 항목입니다.
pipelines.traces.exporters	트레이스 데이터를 전송할 exporters 항목입니다. 예시에서 실제 데이터는 otlp를 통해 전송되고, debug는 디버깅 용도로 사용됩니다.
pipelines.metrics.receivers	메트릭 데이터를 수신하는 receivers 항목입니다.
pipelines.metrics.processors	메트릭 데이터에 적용할 processors 항목입니다.
pipelines.metrics.exporters	메트릭 데이터를 전송할 exporters 항목입니다. 예시에서 실제 데이터는 otlp를 통해 전송되고, debug는 디버깅 용도로 사용됩니다.

부록 A: 문제 해결

본 부록에서는 AnyAPI 게이트웨이 관련 오류에 대한 원인과 해결 방법에 대해 설명합니다.

A.1. FailedGatewayReadsizeBelowZero

원인	SSL 클라이언트쪽에서 연결을 해제 시 발생합니다.
조치사항	openssl s_client 명령을 사용하여 SSL 인증서가 올바른지 확인합니다. <pre>openssl s_client -connect IP:port</pre>

A.2. GatewayAPIAuthenticationFailException

원인	단말 인증에 실패 또는 토큰 값이 다르거나 토큰의 유효기한이 지났을 때 발생합니다.
조치사항	토큰이 정확한지 확인합니다.

A.3. GatewayAPINoMatchingConditionException

원인	클라이언트가 올바른 경로로 호출하지 않았을 때 발생합니다.
조치사항	올바른 경로와 메소드로 호출하였는지 확인합니다. 만약 올바른 경로로 호출하였다면 Admin으로부터 정상 배포가 되었는지 확인하고, DEBUG 로그의 "[ParsingRule] regex match" 로그를 확인하여 제대로 비교되는지 확인합니다.

A.4. GatewayEndpointNotExistException

원인	라우팅할 백엔드 서버가 존재하지 않을 때 발생합니다.
조치사항	백엔드 서버가 제대로 동작하는지 확인하고, 로그 상에서 failover가 발생했는지 확인합니다.

A.5. GatewaySslInitFailException

원인	SSL 인증서의 초기화를 실패했을 때 발생합니다.
조치사항	인증서의 경로 및 비밀번호가 맞는지 확인합니다.

A.6. SSL unknown error

원인	SSL 서버(게이트웨이)에서 발생한 에러로, 잘못된 SSL 요청을 할 경우 발생합니다. (예: https 포트에 http 요청)
조치사항	https 요청을 올바르게 했는지 클라이언트 요청을 확인합니다.

A.7. Invalid HTTP method

원인	HTTP를 파싱할 때 발생하는 에러입니다.
조치사항	HTTP 요청 메시지가 정확한지 확인합니다. error message는 실제 에러가 발생한 메시지를 출력하며 error location은 파싱 문제가 발생한 위치를 말합니다.