



# 외부 서비스 정보 정리 문서



노션 링크

## (1) Google OAuth 서비스

- 서비스명: Google OAuth 2.0 (Google Cloud Console - Credentials)
- 용도
  - 사용자 Google 계정 기반 로그인(소셜 로그인)
- 주요 설정값
  - OAuth 동의 화면: 앱 이름, 지원 이메일, 승인된 도메인
  - OAuth Client ID (Web application)
  - OAuth Client Secret
  - Authorized redirect URI: <https://app.buriburi.monster/callback>
- 활용 위치
  - 프론트엔드: 로그인 페이지
  - 백엔드: OAuth 토큰 검증 및 사용자 정보 조회
  - 로그인 성공 후 JWT 토큰 발급
- 보안/운영
  - Client Secret은 서버 환경 변수로 관리
  - OAuth 동의 화면 검토 필요 시 Google에 제출

## (2) AWS S3, CDN 서비스

- 서비스명: Amazon S3 (Simple Storage Service) / Amazon CloudFront
- 용도

- 사용자 업로드 이미지 및 영상 저장
- 계정
  - AWS 계정 이메일: `Ow0n2x@gmail.com`
- 주요 설정값
  - S3 Bucket name: `buriburi-monster-files`
  - Region:
    - (S3): `ap-northeast-2`
    - (CloudFront): 글로벌
  - CloudFront 대체 도메인 이름: `media.buriburi.monster`
- 활용 위치
  - AI 서버: 웃음 영상 및 썸네일 저장/조회
  - 프론트엔드: 업로드 파일 접근
- 보안/운영
  - 버킷 정책: 공개 접근 제한, 특정 도메인에서만 접근 허용
  - 클라이언트에선 Cloudfront로 접근, 개발 환경에서는 S3 Bucket에 직접 접근

### (3) AWS Lambda

- 서비스명: Amazon S3 (Simple Storage Service)
- 용도
  - GitLab MergeRequest에 대한 등록/댓글 트리거 MatterMost Webhook 전송
- 계정
  - AWS 계정 이메일: `Ow0n2x@gmail.com`
- 코드/환경변수

<code>GITLAB_API_TOKEN</code>	<code>#{YOUR_GITLAB_API_TOKEN}</code>
<code>GITLAB_SECRET_TOKEN</code>	<code>#{YOUR_GITLAB_SECRET_TOKEN}</code>
<code>GITLAB_HOST</code>	<code>https://lab.ssafy.com</code>
<code>MM_WEBHOOK_URL</code>	<code>#{YOUR_MATTERMOST_WEBHOOK_URL}</code>

USER\_MAP

```
{ "dahxtq1": "@dahxtq1", "me_in_u": "@me_in_u",
  "dhnn1536": "@dhnn1536", "phangmin03":
  "@phangmin03", "ryongseong.dev":
  "@ryongseong.dev", "doriconi": "@doriconi" }
```

```
import json
import os
import urllib.request

# --- Lambda 환경 변수 ---
MM_WEBHOOK_URL = os.environ["MM_WEBHOOK_URL"]    # Mattermost 웹훅 URL
SECRET_TOKEN = os.environ["GITLAB_SECRET_TOKEN"] # GitLab 시크릿 토큰

GITLAB_HOST = os.environ.get("GITLAB_HOST") # GitLab 호스트
GITLAB_API_TOKEN = os.environ.get("GITLAB_API_TOKEN") # GitLab API 토큰 (read_api 권한 필요)

USER_MAP_JSON = os.environ.get("USER_MAP") # 유저 맵핑 (GitLab 유저네임: Mattermost 유저네임)
USER_MAP = json.loads(USER_MAP_JSON)

def lambda_handler(event, context):
    """
    AWS Lambda 핸들러 함수 - GitLab 웹훅 이벤트를 처리
    """
    try:
        # GitLab 시크릿 토큰 검증
        if not verify_gitlab_token(event):
            print("Error: Unauthorized Token")
            return {"statusCode": 403, "body": json.dumps({"error": "Unauthorized Token"})}

        # 이벤트 본문 및 타입 파싱
        body = json.loads(event["body"])
        event_type = event["headers"].get("x-gitlab-event", "")
    
```

```

# 처리할 Mattermost 메시지 페이로드 생성
mm_payload = None

if event_type == "Merge Request Hook":
    mm_payload = handle_mr_event(body)
elif event_type == "Note Hook":
    if not GITLAB_API_TOKEN:
        print("Error: GITLAB_API_TOKEN is not set. Cannot process Note Hook.")
        raise Exception("GITLAB_API_TOKEN is not configured")
    mm_payload = handle_note_event(body)
else:
    print(f"Ignoring event type: {event_type}")
    return {"statusCode": 200, "body": json.dumps({"message": "Event ignored"})}

# Mattermost로 메시지 전송
if mm_payload:
    send_to_mattermost(mm_payload)
    return {"statusCode": 200, "body": json.dumps({"message": "Notification sent"})}
else:
    return {"statusCode": 200, "body": json.dumps({"message": "No notification needed"})}

except Exception as e:
    print(f"Error: {str(e)}")
    return {"statusCode": 500, "body": json.dumps({"error": str(e)})}

def verify_gitlab_token(event):
    """
    GitLab 웹훅 요청의 시크릿 토큰을 검증
    """
    received_token = event["headers"].get("x-gitlab-token", "")
    return received_token == SECRET_TOKEN

def get_mm_mention(gitlab_username):

```

```

"""
USER_MAP을 사용해 GitLab 유저네임을 Mattermost 맨션으로 변환
"""

return USER_MAP.get(gitlab_username, f"@{gitlab_username}") # 매
핑이 없으면 일단 @ 붙여서 시도

def handle_mr_event(body):
    """
    [시나리오 1]
    Merge Request 이벤트를 처리 (action: 'open')
    """

    attrs = body.get("object_attributes", {})
    action = attrs.get("action")

    # 1. MR이 처음 열렸을 때만
    if action == "open":
        print("Event: MR Opened")
        mr_title = attrs.get("title", "N/A")
        mr_url = attrs.get("url", "#")
        author_username = body.get("user", {}).get("username", "N/A")
        author_mention = get_mm_mention(author_username)

        # 리뷰어 목록 추출 및 맨션으로 변환
        reviewers = body.get("reviewers", [])
        reviewer_mentions = [get_mm_mention(r["username"]) for r in review
ers]
        mentions_text = ", ".join(reviewer_mentions) if reviewer_mentions e
lse "@channel"

        message = (
            f"----\n"
            f"##### 🚀 {author_mention}님이 새로운 MR을 등록했습니다!\n\n"
            f"**[{mr_title}]({mr_url})**\n\n"
            f"**To. Reviewer** {mentions_text}\n\n"
            f"**24시간 이내** *코드 리뷰를 부탁드립니다.*\n\n"
            f"**MR 페이지 댓글 명령어 안내**\n"
            f"- **`/리뷰완료`**: (리뷰어용) 리뷰 완료 후, MR 작성자에게 확인을 요
청합니다.\n"

```

```

        f"- **`/리뷰다시`** : (작성자용) 리뷰 반영 완료 후, 리뷰어에게 재검토를
        요청합니다.\n"
        f"- **`/리뷰어호출`** : (작성자용) 리뷰어에게 추가 질문/확인을 요청합
        니다.\n\n"
        f"----"
    )

    return {"text": message, "username": "GitLab Bot"}

print(f"Ignoring MR action: {action}")
return None

def get_mr_details_from_api(project_id, mr_iid):
    """
    (신규) GitLab API를 호출하여 MR 상세 정보를 가져옵니다.
    (author_id와 reviewers를 가져오기 위함)
    """
    if not project_id or not mr_iid:
        print("Error: Missing project_id or mr_iid for API call")
        return None

    url = f"{GITLAB_HOST}/api/v4/projects/{project_id}/merge_requests/
{mr_iid}"
    headers = {"PRIVATE-TOKEN": GITLAB_API_TOKEN}

    req = urllib.request.Request(url, headers=headers, method="GET")

    try:
        with urllib.request.urlopen(req) as response:
            if response.status == 200:
                print(f"API call success: fetched MR {mr_iid}")
                return json.loads(response.read().decode("utf-8"))
            else:
                print(f"Error: API call failed with status {response.status}")
                return None
    except Exception as e:
        print(f"Error during API call: {str(e)}")
        return None

```

```

def handle_note_event(body):
    """
    [시나리오 2, 3, 4]
    Note (댓글) 이벤트를 처리
    """

    attrs = body.get("object_attributes", {})
    note_type = attrs.get("noteable_type")

    # 1. MR에 달린 댓글이 아니면 무시
    if note_type != "MergeRequest":
        print(f"Ignoring note type: {note_type}")
        return None

    mr = body.get("merge_request", {})
    project_id = body.get("project", {}).get("id")
    mr_iid = mr.get("iid") # MR의 Internal ID (예: 123)

    mr_details = get_mr_details_from_api(project_id, mr_iid)
    if not mr_details:
        raise Exception(f"Failed to fetch MR details from API for MR iid {mr_iid}")

    mr_title = mr_details.get("title", "N/A")
    mr_url = attrs.get("url", "#") # 댓글 링크
    note_body = attrs.get("note", "")

    # 이벤트 발생자 (댓글 쓴 사람)
    actor_username = body.get("user", {}).get("username", "N/A")
    actor_mention = get_mm_mention(actor_username)

    # MR 작성자
    mr_author_username = mr_details.get("author", {}).get("username", "N/A")
    mr_author_mention = get_mm_mention(mr_author_username)

```

```

# 리뷰어 목록
reviewers = mr_details.get("reviewers", [])
reviewer_usernames = [r["username"] for r in reviewers]
reviewer_mentions = [get_mm_mention(u) for u in reviewer_usernames]

mentions_text = ", ".join(reviewer_mentions) if reviewer_mentions else "@channel"

message_text = ""

# 시나리오 2: 댓글로 리뷰 재요청
if note_body.strip().startswith("/리뷰다시"):
    print("Event: Re-review requested")
    message_text = (
        f"----\n"
        f"##### 👍 {actor_mention}님이 리뷰 재검토를 요청했습니다!\n\n"
        f"**[{mr_title}]({mr_url})**\n\n"
        f"🟡 **To. Reviewer** {mentions_text}\n"
        f"🕒 **24시간 이내** *확인을 부탁드립니다.*\n\n"
        f"----"
    )

# 시나리오 3: MR 작성자가 리뷰어에게 확인 요청
elif note_body.strip().startswith("/리뷰어호출"):
    print("Event: Author requested check from reviewers")
    message_text = (
        f"----\n"
        f"##### 📌 {actor_mention}님이 리뷰어에게 확인을 요청합니다!\n\n"
        f"**[{mr_title}]({mr_url})**\n\n"
        f"🟡 **To. Reviewer** {mentions_text}\n"
        f"🕒 **24시간 이내** *확인을 부탁드립니다.*\n\n"
        f"----"
    )

# 시나리오 4: 리뷰어가 MR 작성자에게 확인 요청
elif note_body.strip().startswith("/리뷰완료"):
    print("Event: Reviewer requested check from author")

```

```

message_text = (
    f"----\n"
    f"##### 📎 {actor_mention}님이 리뷰를 완료했습니다!\n\n"
    f"**[{mr_title}]({mr_url})**\n\n"
    f"👨 **To. Author** {mr_author_mention}\n"
    f"🕒 **24시간 이내** *확인을 부탁드립니다.*\n\n"
    f"----"
)

if message_text:
    return {"text": message_text, "username": "GitLab Bot"}

print(f"Ignoring note: {note_body[:50]}...")
return None

def send_to_mattermost(payload):
    """
    Mattermost 웹훅으로 JSON 페이로드를 전송
    """
    data = json.dumps(payload).encode("utf-8")
    headers = {"Content-Type": "application/json"}

    req = urllib.request.Request(MM_WEBHOOK_URL, data=data, headers=headers, method="POST")

    print(f"Sending payload to Mattermost: {payload}")
    with urllib.request.urlopen(req) as response:
        print(f"Mattermost response status: {response.status}")
    return response.status

```

## (4) Runpod GPU 서버

- 서비스명: Runpod.io GPU 서버
- 용도
  - MediaPipe 기반 영상 처리 및 모델 추론
- 계정

- 이메일: [ryongseong.dev@gmail.com](mailto:ryongseong.dev@gmail.com) (팀원 계정)
- 주요 설정값
  - 사용 플랜
    - GPU 탑재: NVIDIA RTX A4500 (20 GB VRAM)
    - 인스턴스 설정: 12 vCPU, 62GB RAM
    - 스토리지: 80GB SSD
- 활용 위치
  - 백엔드: 영상 처리 API 서버
  - 모델 추론 및 결과 반환
  - MQTT 브로커와 연동