Cloud Computing

A Letter to Myself

-나에게 쓰는 편지

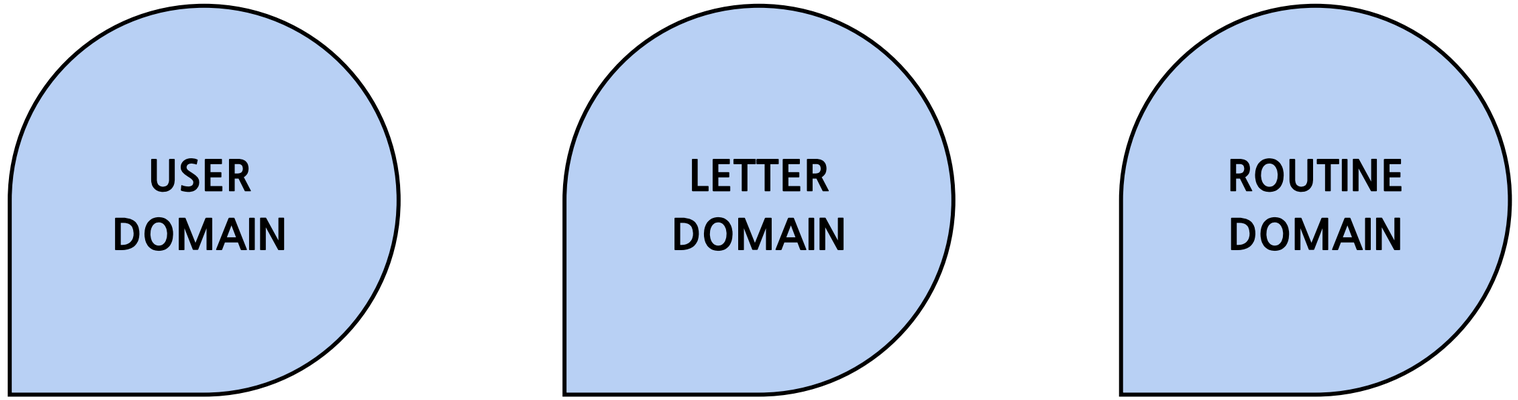
4팀

CONTENTS

1. 프로젝트 개요

모놀리식 구조의 분리

- 기존에 모놀리식으로 운영되었던 웹 서비스를 마이크로 서비스 아키텍처로 리팩토링하기 위한 가장 첫 단계
- 서비스 도메인 기준으로 분리하며, 각 도메인 별로 책임과 역할에 따라 개별 레포지토리로 세분화
- 메인 기능인 user domain, letter domain, routine domain 중심으로 클라우드 기반 아키텍처로 전환



1. 프로젝트 개요

auth-service
사용자 인증 관련 서비스
01

user-service
사용자 프로필 관련 서비스
02

letter-service
편지 작성/열람 서비스
03

letter-storage-service
편지 이미지 저장 서비스
04

routine-service
사용자의 루틴 등록 및 관리 서비스
05

api-gateway

API 라우팅 및 통합

Web-client

사용자 인터페이스

06
scheduler-service
메일 알림 예약 등록 서비스

07
notification-service
리마인더 메일 알림 전송 서비스

08
emotion-ml-service
감정 분석/통계 서비스

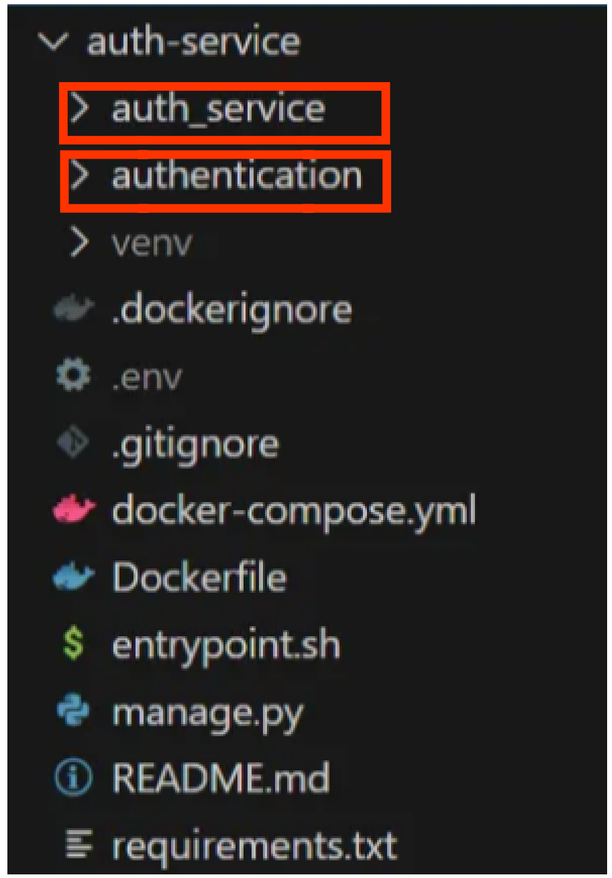
09
emotion-store-service
감정 분석 결과 저장/제공 서비스

10
recommend-service
감정기반 콘텐츠 추천 서비스

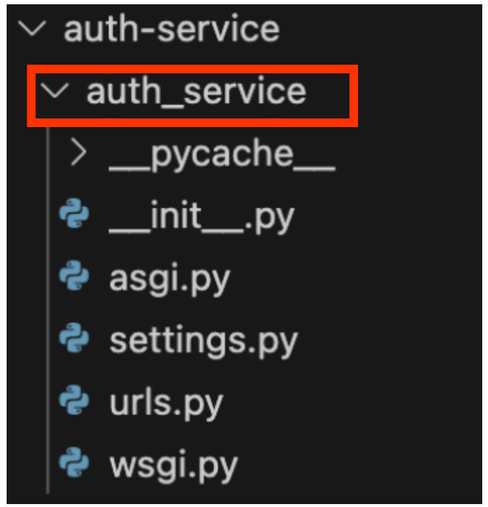
2. 개발 과정 - 폴더 분리 후 로컬 테스트

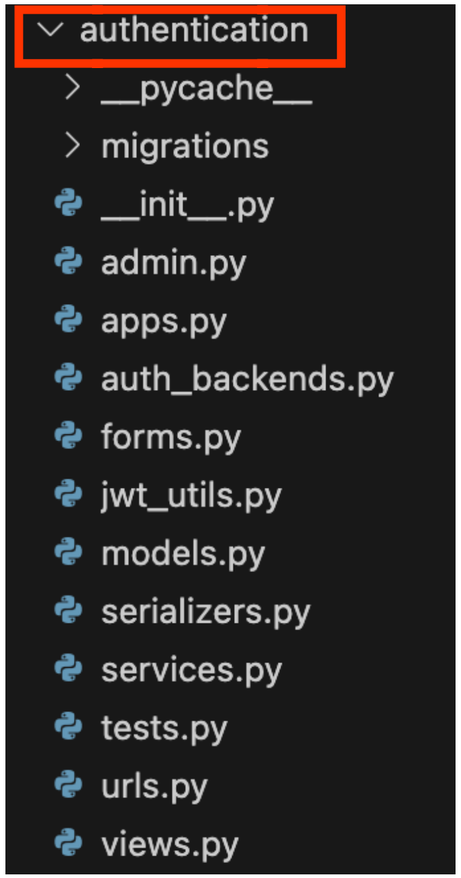
- 각 서비스별 폴더 분리
- 로컬 환경 테스트
  - manage.py runserver
  - 서비스 별 통신 테스트
- 각 서비스별 Dockerfile 작성
- 각 서비스별 컨테이너화

- auth-service 폴더 구조



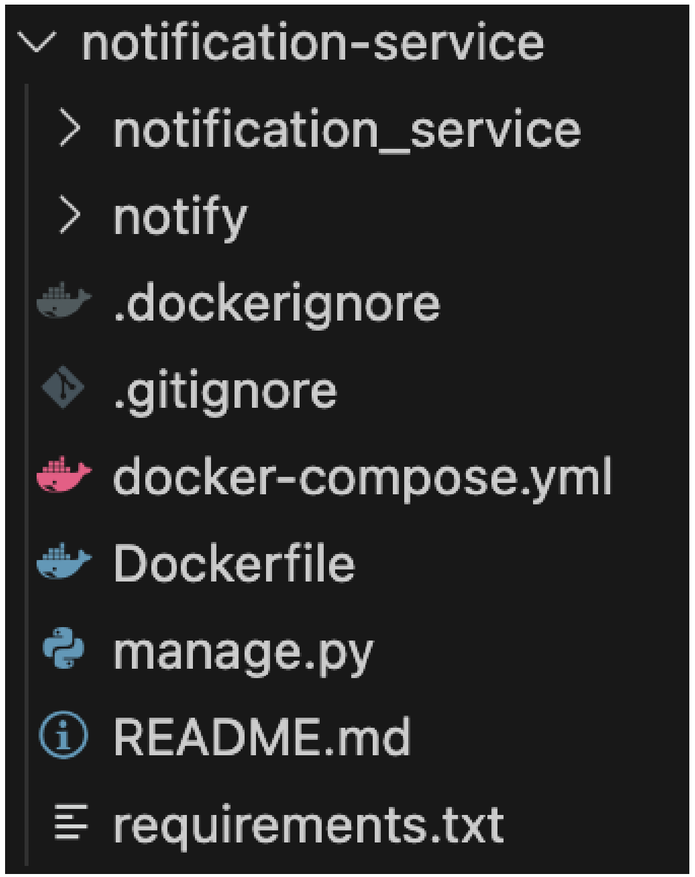
- 백엔드 로직 폴더와 환경 설정 파일 폴더 분리



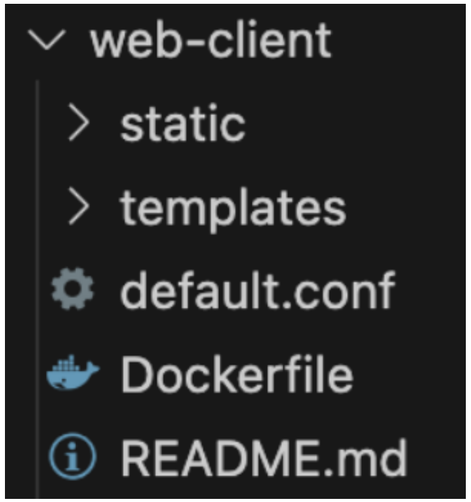


2. 개발 과정 - 폴더 분리 후 로컬 테스트

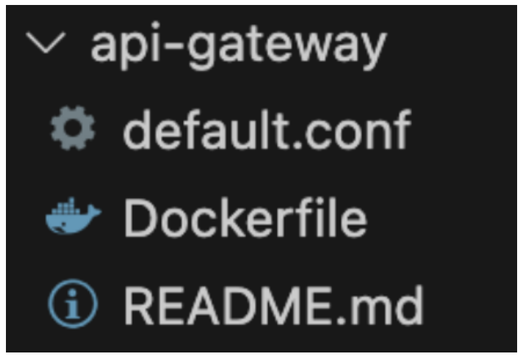
- notification- service



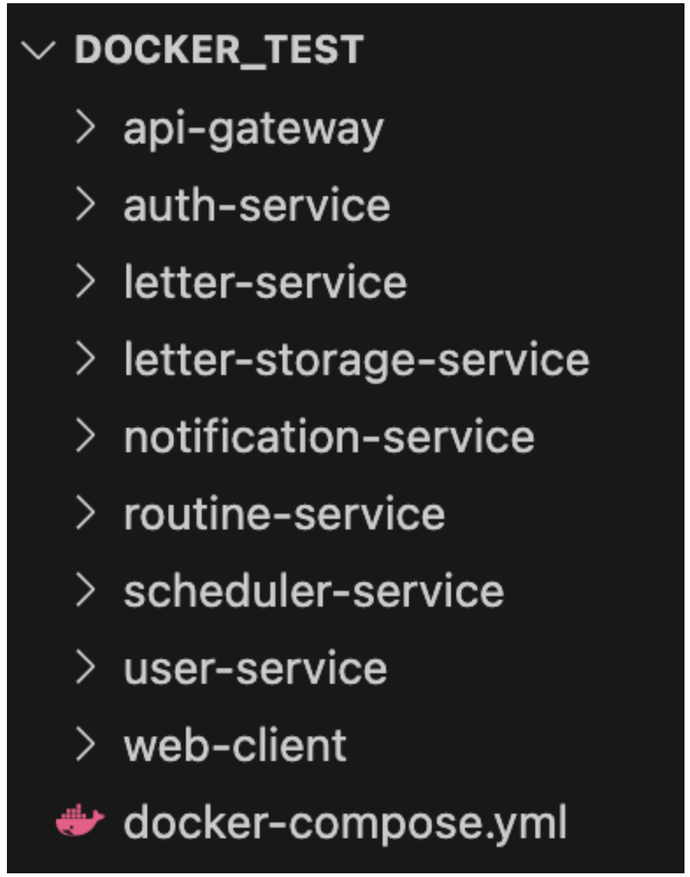
- web-client



- api-gateway



- 전체 폴더 구조



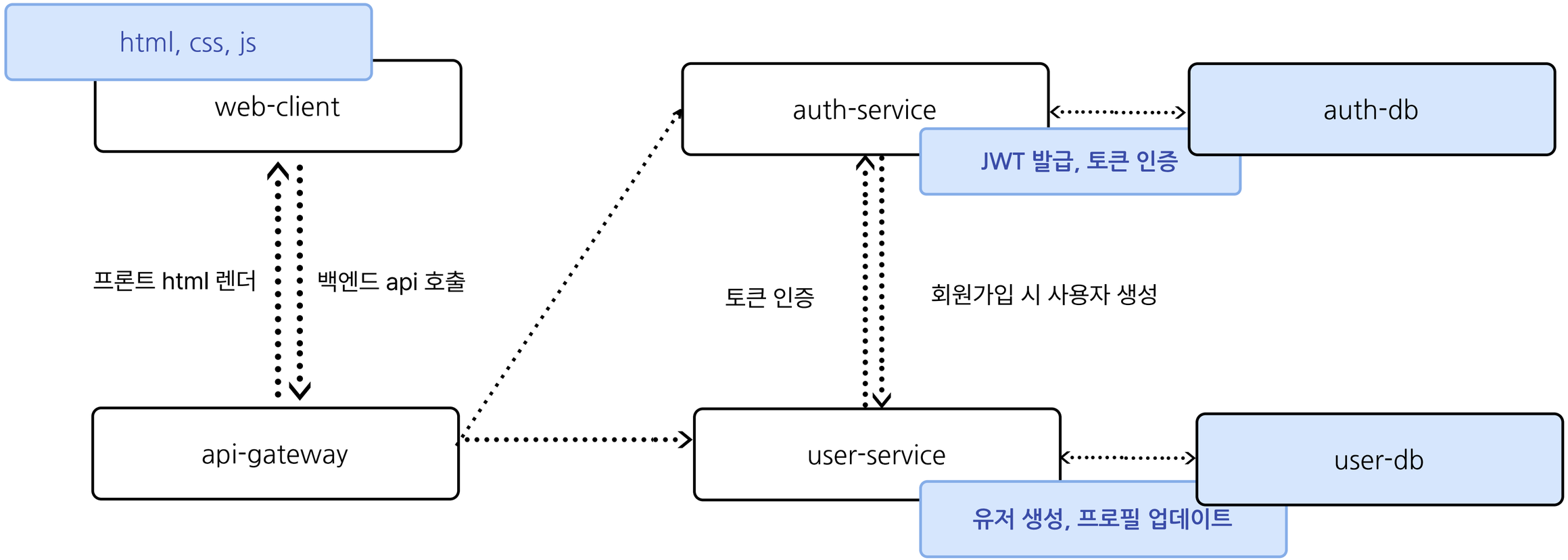
2. 개발 과정 - docker compose 로컬 테스트

- 각 서비스가 독립적으로 작동하는지 테스트하기 위해 서비스 별로 Database 컨테이너 각자 구성
  - auth-db, user-db, routine-db ... 등
  - Database 컨테이너 포트 번호 다르게 설정

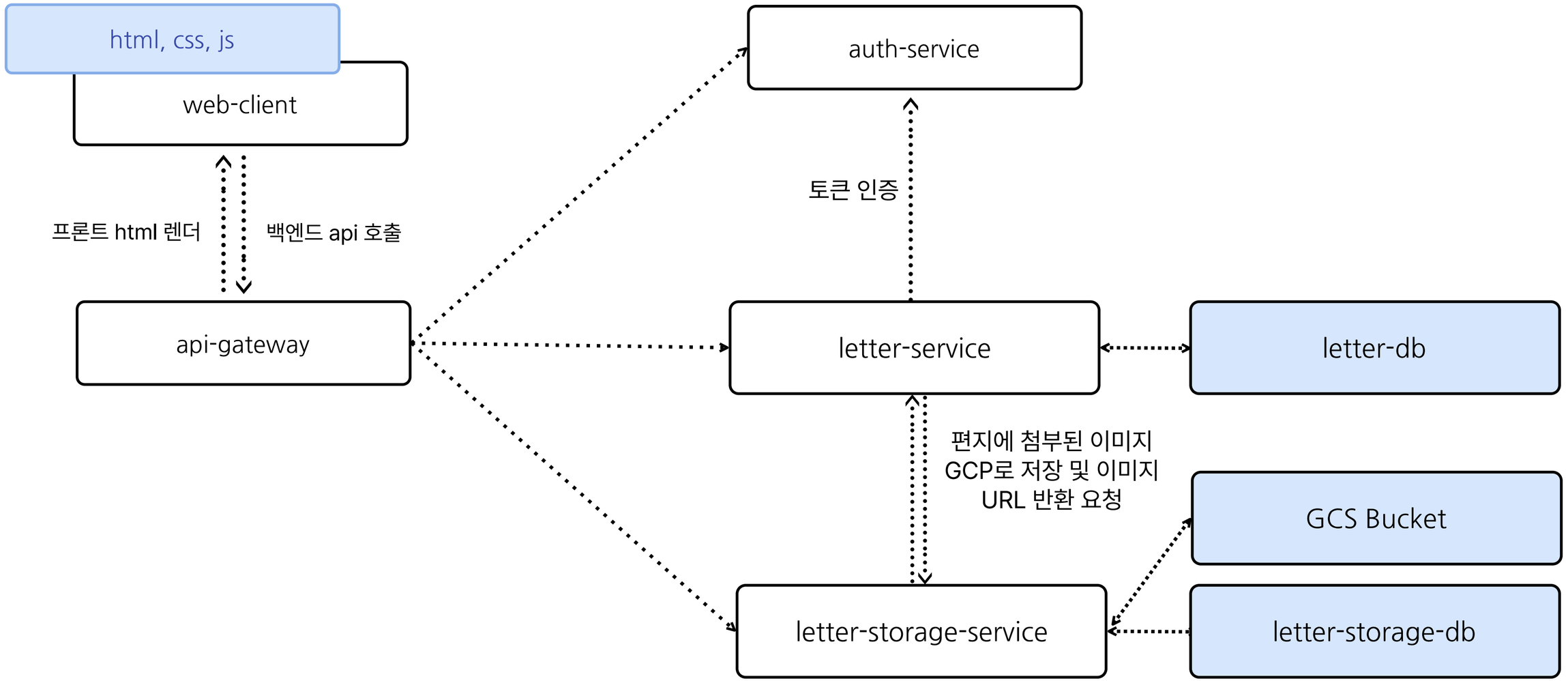
```yaml
auth-db:
  image: postgres:14
  container_name: auth-db
  environment:
    POSTGRES_DB: auth_db
    POSTGRES_USER: auth
    POSTGRES_PASSWORD: auth
  ports:
    - "5432:5432"
  networks:
    - auth-net
  volumes:
    - auth-db-data:/var/lib/postgresql/data
```

```yaml
routine-service-db:
  image: postgres:14
  container_name: routine-service-db
  restart: always
  env_file:
    - .env
  environment:
    POSTGRES_DB: ${DB_NAME}
    POSTGRES_USER: ${DB_USER}
    POSTGRES_PASSWORD: ${DB_PASSWORD}
  ports:
    - 5435:5432
  volumes:
    - routine-service-db-data:/var/lib/postgresql/data
  healthcheck:
    test: ["CMD", "pg_isready", "-U", "postgres"]
    interval: 10s
    timeout: 5s
    retries: 5
  networks:
  - schedule-internal-net
  - auth-share-net
```

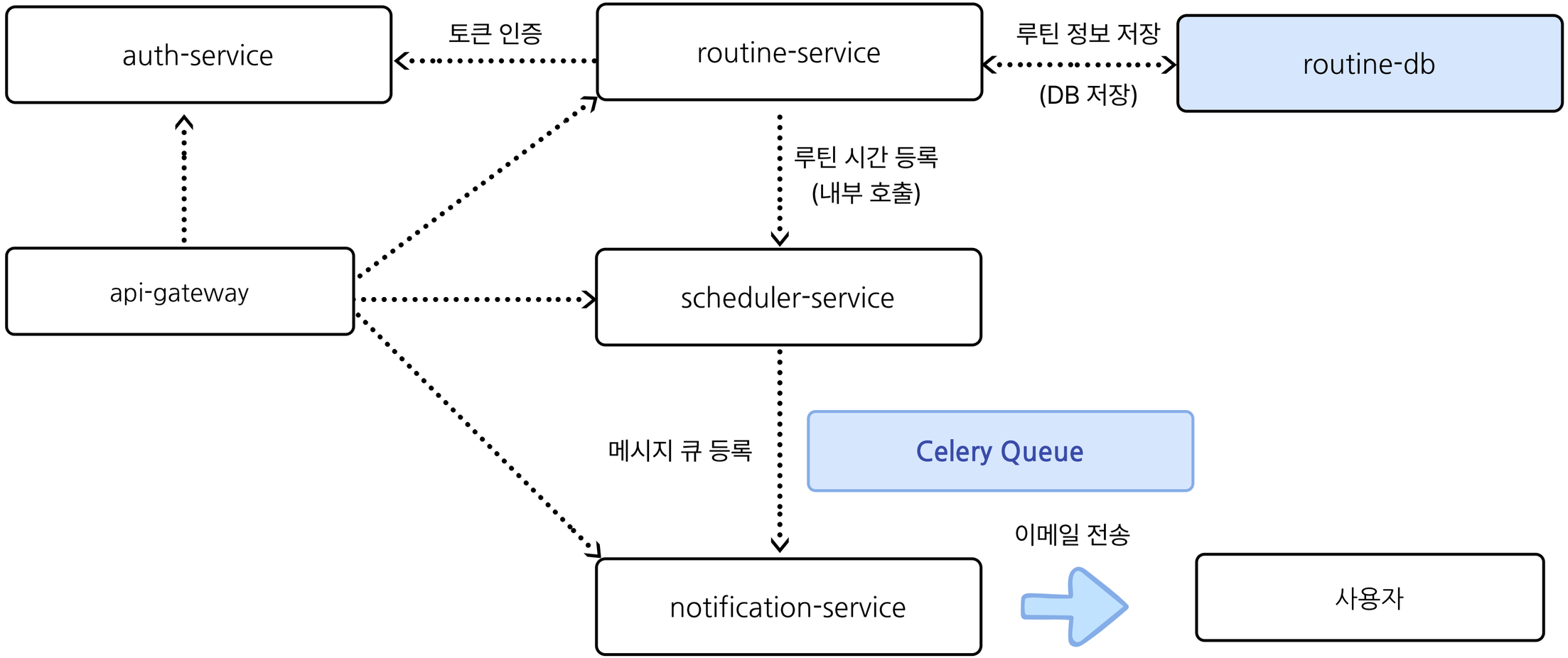
3. 마이크로 서비스 통신 - USER DOMAIN



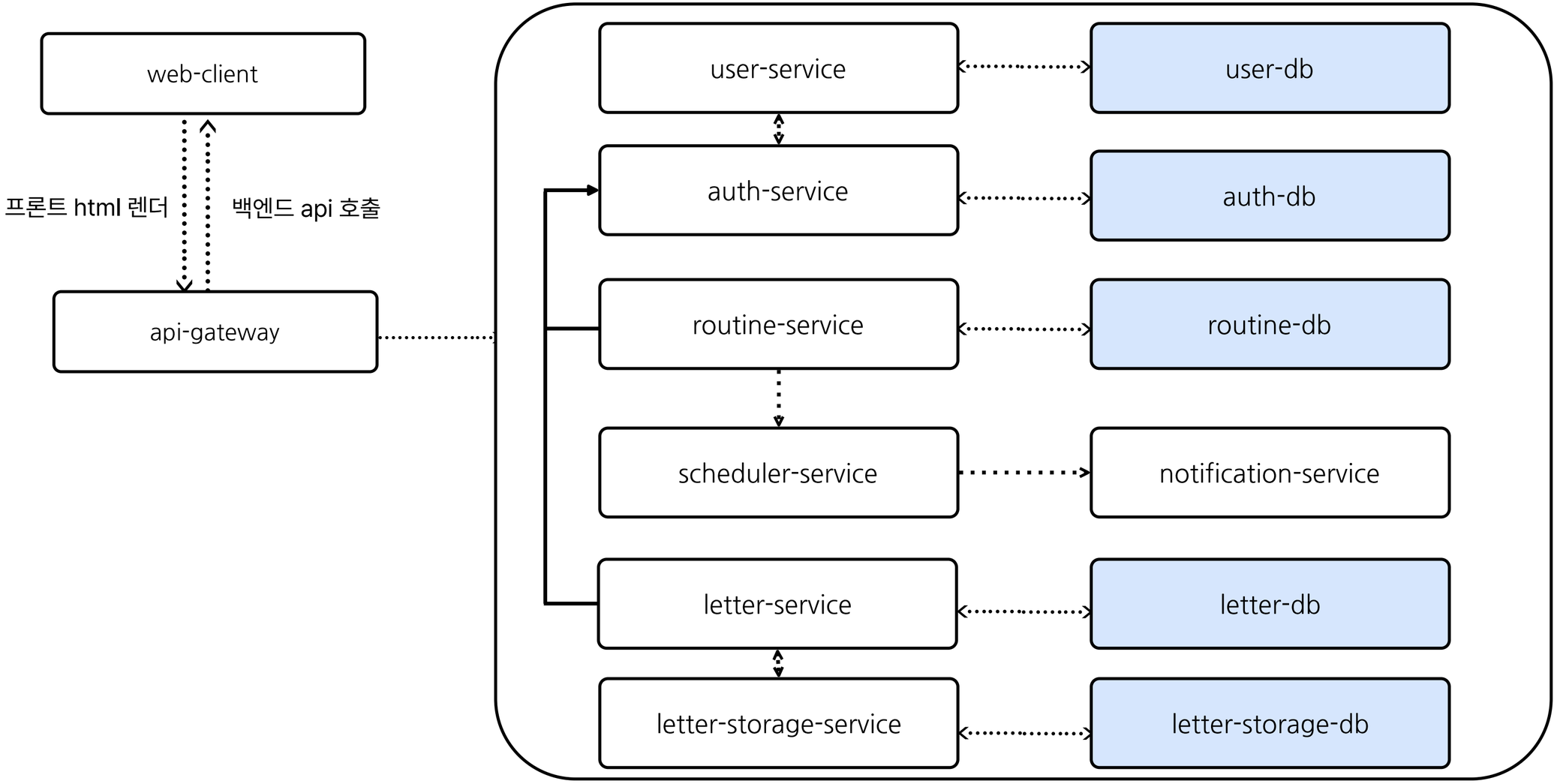
3. 마이크로 서비스 통신 - LETTER DOMAIN



3. 마이크로 서비스 통신 - SCHEDULER DOMAIN



3. 마이크로 서비스 통신 - 전체 구조



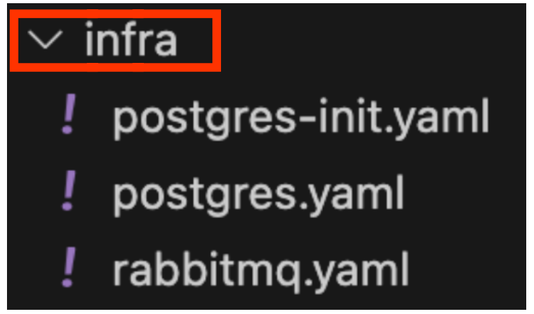
4. 운영 구조 개선 - Database

- DB 구조 변경
  - 초기
    - 서비스마다 DB 컨테이너 별도 생성
  - 문제
    - 컨테이너 수 증가하며

      유지보수의 편의성 및 불필요한 리소스 사용
  - 피드백
    - DB 컨테이너 분리
    - PostgreSQL 1개로 통합
  - 수정
    - 쿠버네티스 yaml 파일 작성
    - PostgreSQL 1개로 DB 분리 진행

- 쿠버네티스 파일 통합 폴더

- Database, RabbitMQ 파일



- postgres-init.yaml

```
! postgres-init.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-init-script
data:
  init.sql: |
    CREATE DATABASE auth_db;
    CREATE DATABASE user-db;
    CREATE DATABASE routine_db;
    CREATE DATABASE letters_db;
    CREATE DATABASE letter_storage_db;
```

4. 운영 구조 개선 - PostgreSQL, RabbitMQ

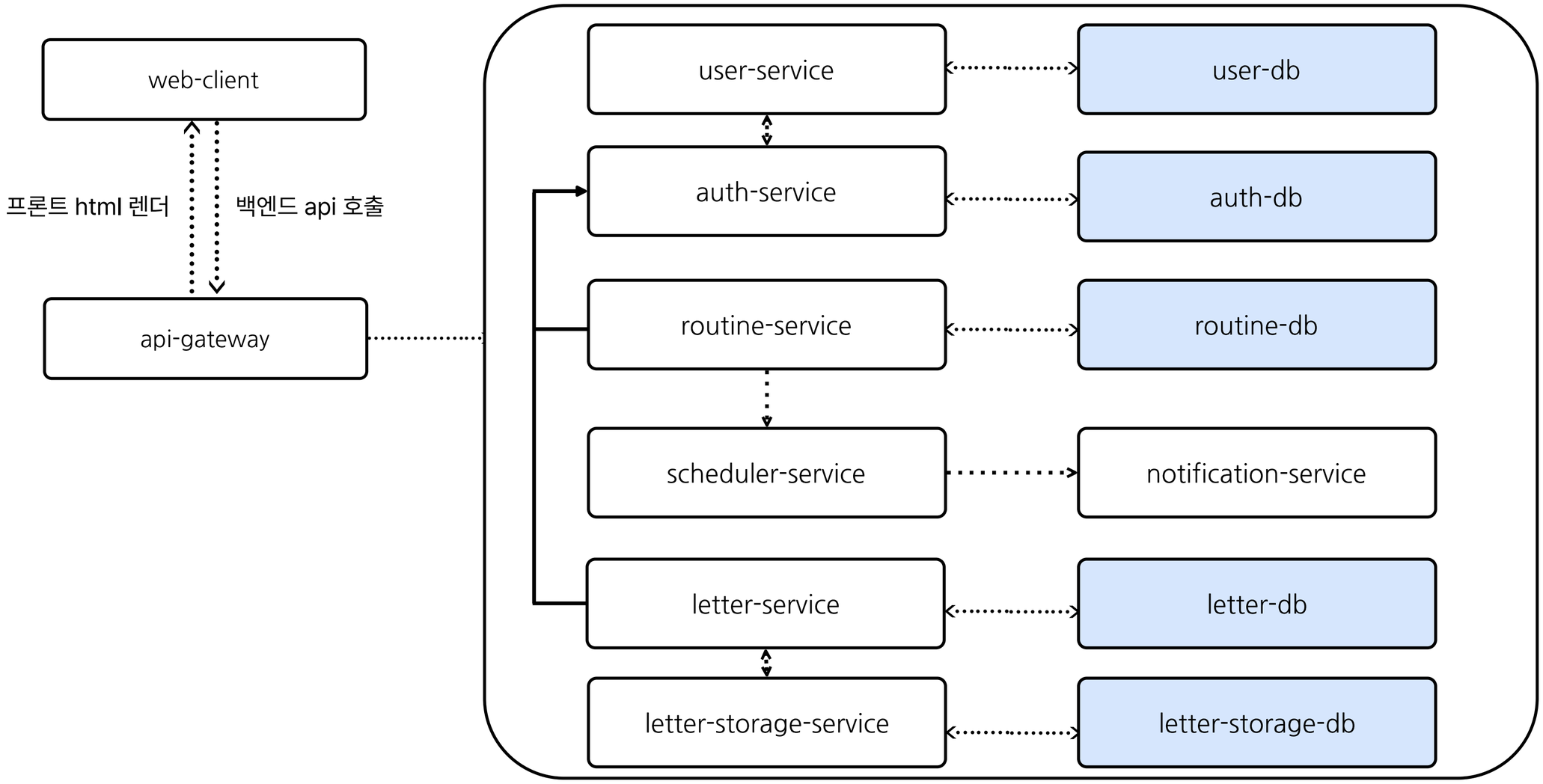
- postgres.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:14
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_USER
              value: postgres
            - name: POSTGRES_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: postgres-secret
                  key: password
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: pgdata
            - name: init-script-volume
              mountPath: /docker-entrypoint-initdb.d
      volumes:
        - name: pgdata
          emptyDir: {}
        - name: init-script-volume
          configMap:
            name: postgres-init-script
---
apiVersion: v1
kind: Secret
metadata:
  name: postgres-secret
type: Opaque
stringData:
```

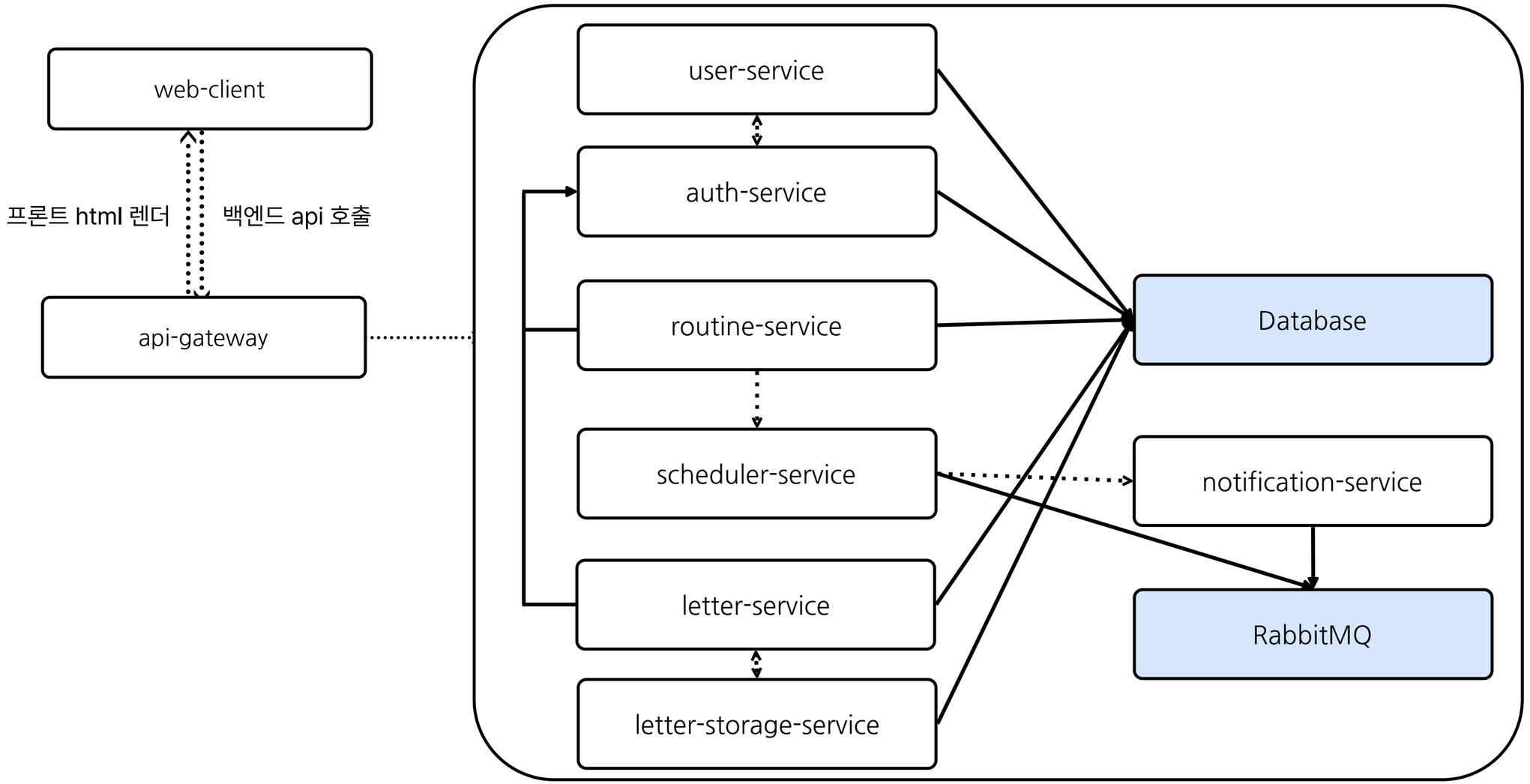
- rabbitmq.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rabbitmq
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rabbitmq
  template:
    metadata:
      labels:
        app: rabbitmq
    spec:
      containers:
        - name: rabbitmq
          image: rabbitmq:3-management
          ports:
            - containerPort: 5672
            - containerPort: 15672
```

4. 운영 구조 개선 - before



4. 운영 구조 개선 - after



4. 운영 구조 개선 - 통합 관리 repository

- 쿠버네티스 통합 관리를 위한 repository 생성
- 각 도메인 별 폴더와 infra 폴더
- infra 폴더
  - 1개의 컨테이너만 구성하는 PostgreSQL 과 RabbitMQ을 관리
  - 통신 편의성과 보안 고려
  - 쿠버네티스 클러스터 안에서 함께 배포
  - 네트워크 설정 간소화

Database 격리는 컨테이너 단위가 아닌, 논리적 Database/schema 단위로 설계

5. 쿠버네티스 오브젝트 - Deployment

- deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: routine-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: routine-service
  template:
    metadata:
      labels:
        app: routine-service
    spec:
      containers:
        - name: routine-service
          image: nye0817/al2m-routine-service
          ports:
            - containerPort: 8003
```

- docker compose의 한계
  - docker compse로 테스트
  - 네트워크 문제,

    의존 컨테이너 등

    동기화 문제 발생

- configmap 과 secret 사용

```yaml
envFrom:
  - configMapRef:
      name: routine-config
  - secretRef:
      name: routine-secret
```

- health check 사용

```yaml
livenessProbe:
  httpGet:
    path: /health/
    port: 8003
  initialDelaySeconds: 10
  periodSeconds: 10
readinessProbe:
  httpGet:
    path: /health/
    port: 8003
  initialDelaySeconds: 5
  periodSeconds: 5
```

```
[09/Jun/2025 03:54:45] "GET /health/ HTTP/1.1" 200 16
[09/Jun/2025 03:54:49] "GET /health/ HTTP/1.1" 200 16
[09/Jun/2025 03:54:54] "GET /health/ HTTP/1.1" 200 16
[09/Jun/2025 03:54:55] "GET /health/ HTTP/1.1" 200 16
[09/Jun/2025 03:54:59] "GET /health/ HTTP/1.1" 200 16
[09/Jun/2025 03:55:00] "GET /api/routines/today/ HTTP/1.1" 200 2
[09/Jun/2025 03:55:04] "GET /health/ HTTP/1.1" 200 16
[09/Jun/2025 03:55:05] "GET /health/ HTTP/1.1" 200 16
```

5. 쿠버네티스 오브젝트 - Deployment (health check)

- deployment.yaml

```yaml
livenessProbe:
  httpGet:
    path: /health/
    port: 8003
  initialDelaySeconds: 10
  periodSeconds: 10
readinessProbe:
  httpGet:
    path: /health/
    port: 8003
  initialDelaySeconds: 5
  periodSeconds: 5
```

- urls.py 헬스체크 endpoint 추가

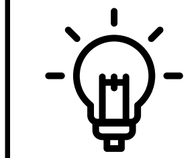
```python
app_name = 'routines'

urlpatterns = [
    path('admin/', admin.site.urls),
    path("", TemplateView.as_view(template_name="comm
    path("routine/", include("routine.urls")),
    path("api/routines/", include("routine.urls")),
    path("health/", health_check),
]
```

- views.py 헬스체크 구현

```python
@require_GET
def health_check(request):
    return JsonResponse({"status": "ok"})
```

5. 쿠버네티스 오브젝트 - Deployment(ConfigMap 과 Secret)



- 기존 로컬 개발 환경에서 env 파일에 비밀번호, 인증키와 같은 민감한 정보를 설정
- 도커와 쿠버네티스를 도입하면서 중요한 데이터가 도커 이미지에 포함되지 않도록 수정
- .dockerignore에 .env 등을 추가
- Dockerfile에서 런타임 때 환경 변수 값이 설정되도록 수정

- .dockerignore

```
__pycache__/
*.pyc
*.pyo
*.pyd
*.sqlite3
.env
.git
.gitignore
docker-compose.yml
```

- Dockerfile

```
# 환경 변수 설정을 위한 build arguments
ARG DB_NAME
ARG DB_USER
ARG DB_PASSWORD
ARG DB_HOST
ARG DB_PORT
ARG SECRET_KEY

# 런타임 환경 변수 설정
ENV DB_NAME=$DB_NAME \
    DB_USER=$DB_USER \
    DB_PASSWORD=$DB_PASSWORD \
    DB_HOST=$DB_HOST \
    DB_PORT=$DB_PORT \
    SECRET_KEY=$SECRET_KEY
```

5. 쿠버네티스 오브젝트 - Deployment(ConfigMap 과 Secret)

- deployment.yaml

```yaml
envFrom:
  - configMapRef:
      name: routine-config
  - secretRef:
      name: routine-secret
```

- Deployment.yaml 에서 configmap과 secret 활용하도록 설정
- 설정 정보를 한번에 사용하기 위해 envForm으로 등록

- configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: routine-config
data:
  DEBUG: "True"
  DB_NAME: routine_db
  DB_USER: postgres
  DB_HOST: postgres-service
  DB_PORT: "5432"
  PORT: "8003"
  RABBITMQ_HOST: rabbitmq-service
  RABBITMQ_PORT: "5672"
```

- Database 이름, RabbitMQ 이름
- 포트 번호, 사용자 이름

- secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: routine-secret
type: Opaque
stringData:
  SECRET_KEY: django-insec
  EMAIL_HOST_PASSWORD: vtz
  OPENAI_API_KEY: sk-proj-
  DB_PASSWORD: sksk0877
```

- 비밀번호
- 인증키

5. 쿠버네티스 오브젝트 - Service

- ClusterIP
  - 외부 노출 X
  - 내부 서비스들 간의 통신

```yaml
apiVersion: v1
kind: Service
metadata:
  name: routine-service
spec:
  selector:
    app: routine-service
  ports:
    - port: 8003
      targetPort: 8003
  type: ClusterIP
```

- LoadBalancer
  - api-gateway 서비스만 외부 노출
  - api-gateway를 통해 사용자 접근

```yaml
apiVersion: v1
kind: Service
metadata:
  name: api-gateway-service
spec:
  selector:
    app: api-gateway
  ports:
  - port: 80
    targetPort: 80
  type: LoadBalancer
```

5. 쿠버네티스 오브젝트 - Service

- 사용자는 외부에서 EXTERNAL-IP를 입력하여 api-gateway를 통해 서비스 내부로 진입한다.
- api-gateway는 .conf 파일에 등록된 url에 따라 web-client와 그 외 서비스로 전달한다.
- api-gateway만 LoadBalancer로 등록하여 외부에 노출시킨다.
- 다른 모든 서비스들은 ClusterIP로 등록하여 내부 서비스들끼리만 통신한다.

```
NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)              AGE
api-gateway-service    LoadBalancer   34.118.237.43    34.47.123.86   80:30322/TCP         18h
auth-service           ClusterIP      34.118.230.42    <none>         8001/TCP             22h
kubernetes             ClusterIP      34.118.224.1     <none>         443/TCP              22h
letter-service         ClusterIP      34.118.237.170   <none>         8006/TCP             21h
notification-service   ClusterIP      34.118.239.184   <none>         8005/TCP             22h
postgres-service       ClusterIP      34.118.235.88    <none>         5432/TCP             22h
rabbitmq-service       ClusterIP      34.118.230.71    <none>         5672/TCP,15672/TCP   22h
routine-service        ClusterIP      34.118.237.65    <none>         8003/TCP             22h
scheduler-service      ClusterIP      34.118.226.164   <none>         8004/TCP             22h
user-service           ClusterIP      34.118.227.167   <none>         8002/TCP             22h
web-client-service     ClusterIP      34.118.227.106   <none>         80/TCP               18h
```

6. 트러블 슈팅 (1) - GCS 버킷 인증 문제

- GCS 버킷과 연결된 letter-storage-service에서 인증과 관련된 GCP_SA_KEY을 secret.yaml에 넣어주었음에도, 환경변수를 찾지 못하는 문제 발생

```
$ kubectl logs letter-storage-service-6b67f69bb5-knwr5
[06/Jun/2025 16:17:48] "GET /health/ HTTP/1.1" 400 143
[06/Jun/2025 16:17:51] "GET /health/ HTTP/1.1" 400 143
[06/Jun/2025 16:17:53] "GET /health/ HTTP/1.1" 400 143
[06/Jun/2025 16:17:58] "GET /health/ HTTP/1.1" 400 143
[06/Jun/2025 16:18:01] "GET /health/ HTTP/1.1" 400 143
[06/Jun/2025 16:18:03] "GET /health/ HTTP/1.1" 400 143
[06/Jun/2025 16:18:08] "GET /health/ HTTP/1.1" 400 143
[06/Jun/2025 16:18:11] "GET /health/ HTTP/1.1" 400 143
[06/Jun/2025 16:18:11] "GET /health/ HTTP/1.1" 400 143
GOOGLE_APPLICATION_CREDENTIALS_JSON 환경변수가 존재하지 않습니다.
```

```
stringData:
  SECRET_KEY: django-insecure-p+u01mq5w&^l+cubq^78#ont(=or5@1oi8hno%0f6rgf#$
  LETTER_STORAGE_DB_PASSWORD: sksk0877
  BUCKET_NAME: a_letter_to_myself_letter_images
  GCP_SA_KEY: |
    {
      "type": "service_account",
      "project_id": "a-letter-to-myself-460117",
      "private_key_id": "4018fda8ec6050f13c25db3fb69859760c620604",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvQIBADANBgkqhkiG9w0BA
      "client_email": "letter-images-storage-access@a-letter-to-myself-46011
      "client_id": "118323821495326619980",
      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
      "token_uri": "https://oauth2.googleapis.com/token",
      "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/c
      "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/
      "universe_domain": "googleapis.com"
    }
```

k8s config > letter-storage > secret.yaml

6. 트러블 슈팅 (1) - GCS 버킷 인증 문제

- 원인
  - os.getenv('GCP_SA_KEY')을 통해 환경변수를 불러왔는데,
    GCP_SDK에서는 경로 기반의 GOOGLE_APPLICATION_CREDENTIALS요구
  - 현재 GCP_SA_KEY는 JSON 문자열 형태로 환경변수에 넣어주었기 때문에 문제 발생
- 해결
  - entrypoint.sh에 GCP_SA_KEY 문자열을 임시 파일로 저장하고 경로를 지정해준 뒤, Dockerfile에서 entrypoint를 지정해주어 Google Cloud SDK에서 인식할 수 있도록 해주었다.
  - k8s Deployment에서 Docker 이미지가 실행되면, Dockerfile에서 지정한 entrypoint.sh가 자동 실행되어 초기 설정을 마치고 앱을 시작

6. 트러블 슈팅 (1) - GCS 버킷 인증 문제

```
#!/bin/sh

echo "$GCP_SA_KEY" > /tmp/gcp-key.json
export GOOGLE_APPLICATION_CREDENTIALS="/tmp/gcp-key.json"

exec "$@"
```

letter-storage-service > entrypoint.sh

```
# 앱 코드 및 entrypoint 복사
COPY . .
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

# entrypoint 지정
ENTRYPOINT ["/entrypoint.sh"]
```

letter-storage-service > Dockerfile

6. 트러블 슈팅 (2) - RabbitMQ 컨테이너 unhealthy 상태 지속 문제

- 원인
  - 기동은 되었지만 healthcheck 기준이 너무 엄격해서 실패
- 해결
  - healthcheck 설정을 완화 (interval, timeout, retries 조정)
  - 처음 기동 시 몇 초 안에 ping 못 찍으면 그냥 죽여버리는 것이기 때문

```
on:
rabbitmq                | 2025-06-03 11:40:15.894861+00:00 [warnin
g] <0.723.0>     "deprecated_features.permit.management_metrics_
collection = true"
rabbitmq                | 2025-06-03 11:40:15.894861+00:00 [warnin
g] <0.723.0> To test RabbitMQ as if the feature was removed, set
 this in your configuration:
rabbitmq                | 2025-06-03 11:40:15.894861+00:00 [warnin
g] <0.723.0>     "deprecated_features.permit.management_metrics_
collection = false"
Gracefully stopping... (press Ctrl+C again to force)
[+] Stopping 3/0
 ✓ Container notification_worker    Stopped                  0.0s
 ✓ Container notification_service   Stopped                  0.0s
 - Container auth-service           Stopping                 0.0s
 ✓ Container scheduler_worker       Stopped                  0.0s
 - Container scheduler_service      Stopping                 0.0s
 - Container scheduler_beat         Stopping                 0.0s
dependency failed to start: container rabbitmq is unhealthy
```

6. 트러블 슈팅 (3) - scheduler-service 파드 개수 문제

- scheduler 서비스는 http 요청받는 파드와 woker 파드, beat 파드 3개로 구성되어 있어야 하지만 http 요청을 받는 파드가 2개가 실행되는 문제 발생
- ReplicaSet이 새로 갱신되면서 이전 Pod을 못 죽이고 있어서 중복 실행되고 있는 상태
- 이미지 업데이트 등으로 scheduler-service의 Deployment가 롤링 업데이트 중에 새 Pod 실행이 실패(CrashLoopBackOff)하면서 이전 Pod도 못 지우고 유지 중

```
notification-service-6cdcdf869f-8jrmq   0/1     Running            5 (34s ago)     3m55s
notification-service-6fff974644-6wllc   0/1     Running            1 (4s ago)      45s
notification-worker-d85c9bb6b-p2mgf     1/1     Running            0               47m
postgres-864854669-h2ldg                1/1     Running            0               47m
rabbitmq-86587c58d5-4jcfm               1/1     Running            0               47m
routine-service-588d9f5f8b-9wsvp        0/1     Running            5 (35s ago)     3m56s
routine-service-8458f45ddd-ln8pz        0/1     Running            1 (16s ago)     47s
scheduler-beat-d7bdc8946-7rwbz          0/1     CrashLoopBackOff   6 (3m36s ago)   9m29s
scheduler-service-66d77d657-2zgx5       0/1     CrashLoopBackOff   2 (20s ago)     46s
scheduler-service-7b696796b7-8q22r      0/1     CrashLoopBackOff   5 (34s ago)     3m55s
scheduler-worker-876c8c49d-mpl8k        0/1     CrashLoopBackOff   6 (3m40s ago)   9m30s
user-service-5f6bb489b9-tpns7           0/1     Running            1 (7s ago)      47s
user-service-64856d6986-mbw8n           0/1     Running            5 (36s ago)     3m57s
```

6. 트러블 슈팅 (3) - scheduler-service 파드 개수 문제

- 현재 scheduler 관련 Deployment 목록 확인
  - kubectl get deployments | findstr scheduler
- ReplicaSet 확인하여 중복 pod의 원인 찾기
  - kubectl get rs | findstr scheduler
  - ReplicaSet 9개 있는 것 확인
- Pod 0개인 유령 ReplicaSet과 오래된 ReplicaSet 삭제

```
C:\Users\nameo\AppData\Local\Google\Cloud SDK\k8s-configs>kubectl get rs | findstr scheduler-service
scheduler-service-55ddbcc56f     0         0         0       19m
scheduler-service-5d86ffc7cc     0         0         0       76m
scheduler-service-5f47967cdd     0         0         0       69m
scheduler-service-66d77d657      1         1         0       11m
scheduler-service-6b959dd4f5     0         0         0       56m
scheduler-service-6bf6df6dc9     0         0         0       42m
scheduler-service-7b4bc4bc65     0         0         0       19m
scheduler-service-7b696796b7     1         1         0       14m
scheduler-service-df7859b79      0         0         0       76m
```

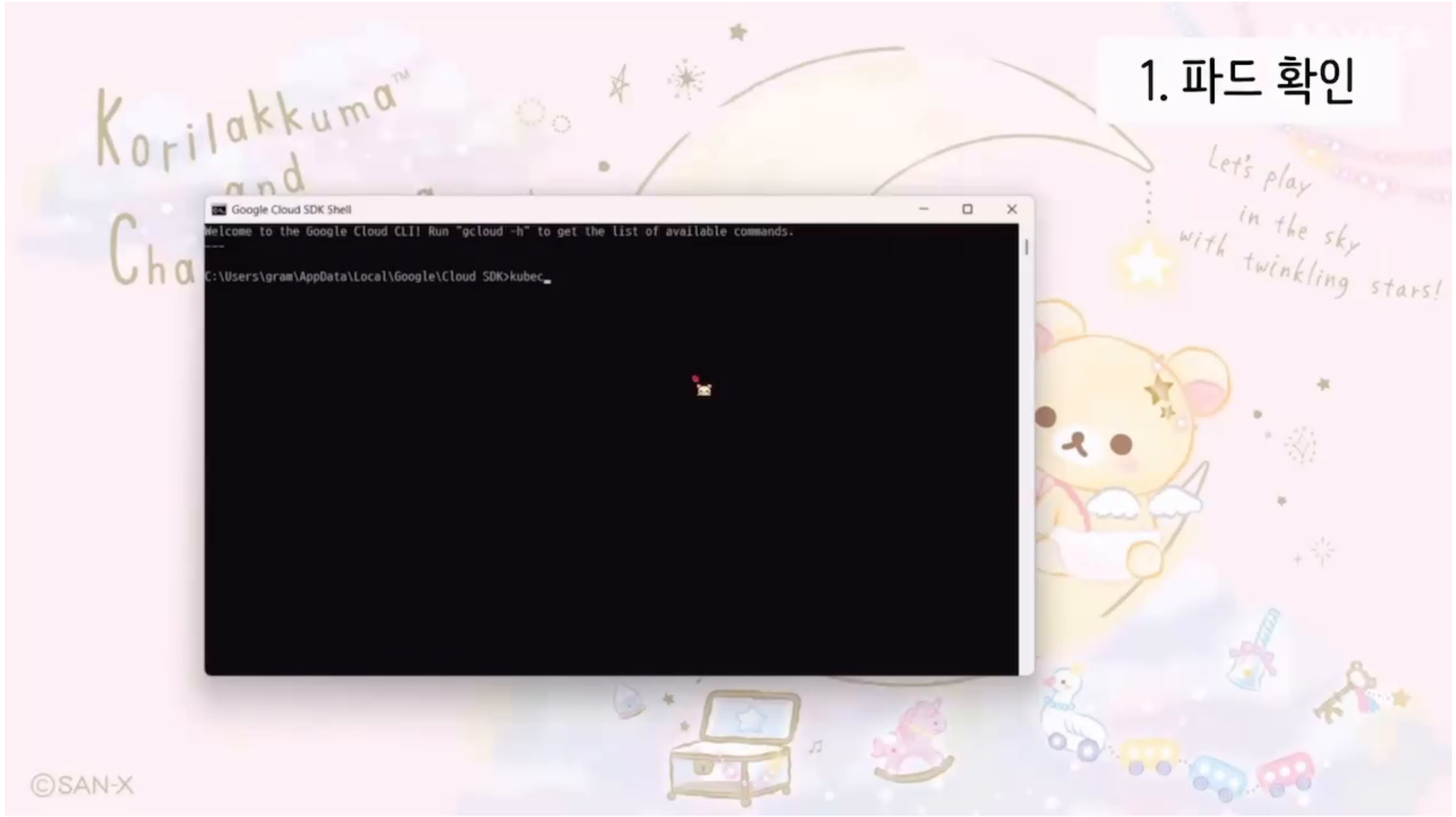
6. 트러블 슈팅 (4) - health check 실패 문제

- /health/ 엔드포인트 호출 시 500 에러 발생으로 pod가 ready되지 않음
- 원인
  - 모놀리식 구조에서는 인증 관련 서비스들이 한 프로젝트에 있어 DRF 인증 기능이 자연스럽게 작동
  - 마이크로서비스 구조에서는 인증 구성요소가 분리되어 있고, DRF의 @api_view 데코레이터는 request.user 접근 시 인증 시도가 오류로 이어짐
- 해결
  - health check는 사용자 인증이 필요하지 않으므로, DRF 인증이 필요없는 view는 Django 기본 @require_GET view 사용으로 해결

```
@api_view(['GET'])
def health_check(request):
    return JsonResponse({"status": "ok"})
```

```
@require_GET
def health_check(request):
    return JsonResponse({"status": "ok"})
```

데모 영상


1. 파드 확인
Google Cloud SDK Shell
Welcome to the Google Cloud CLI! Run "gcloud -h" to get the list of available commands.
C:\Users\gram\AppData\Local\Google\Cloud SDK>kubec
Let's play in the sky with twinkling stars!
©SAN-X

THANK YOU!
감사합니다!

클라우트 컴퓨팅 4팀