가족 도착 알리미

Table of contents

서비스 시나리오

기능적 요구사항

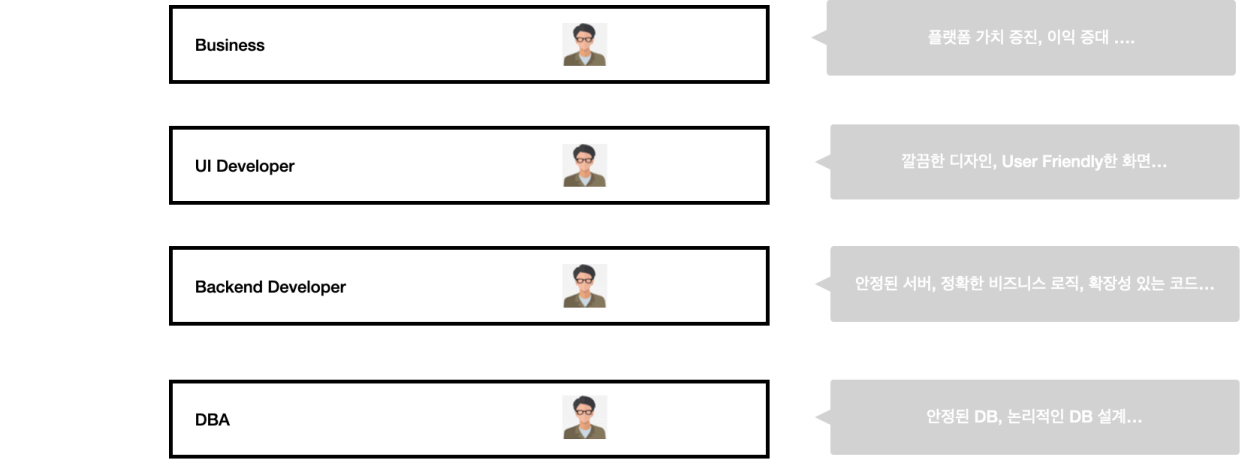
1. 고객 가족구성원을 등록/삭제/조회한다.
2. 원격지에서 가족구성원이 이동을 시작하면 웹으로 전달된다.
3. 가족구성원이 원격지에서 출발한다.
4. 가족구성원이 이동을 취소할 수 있다.

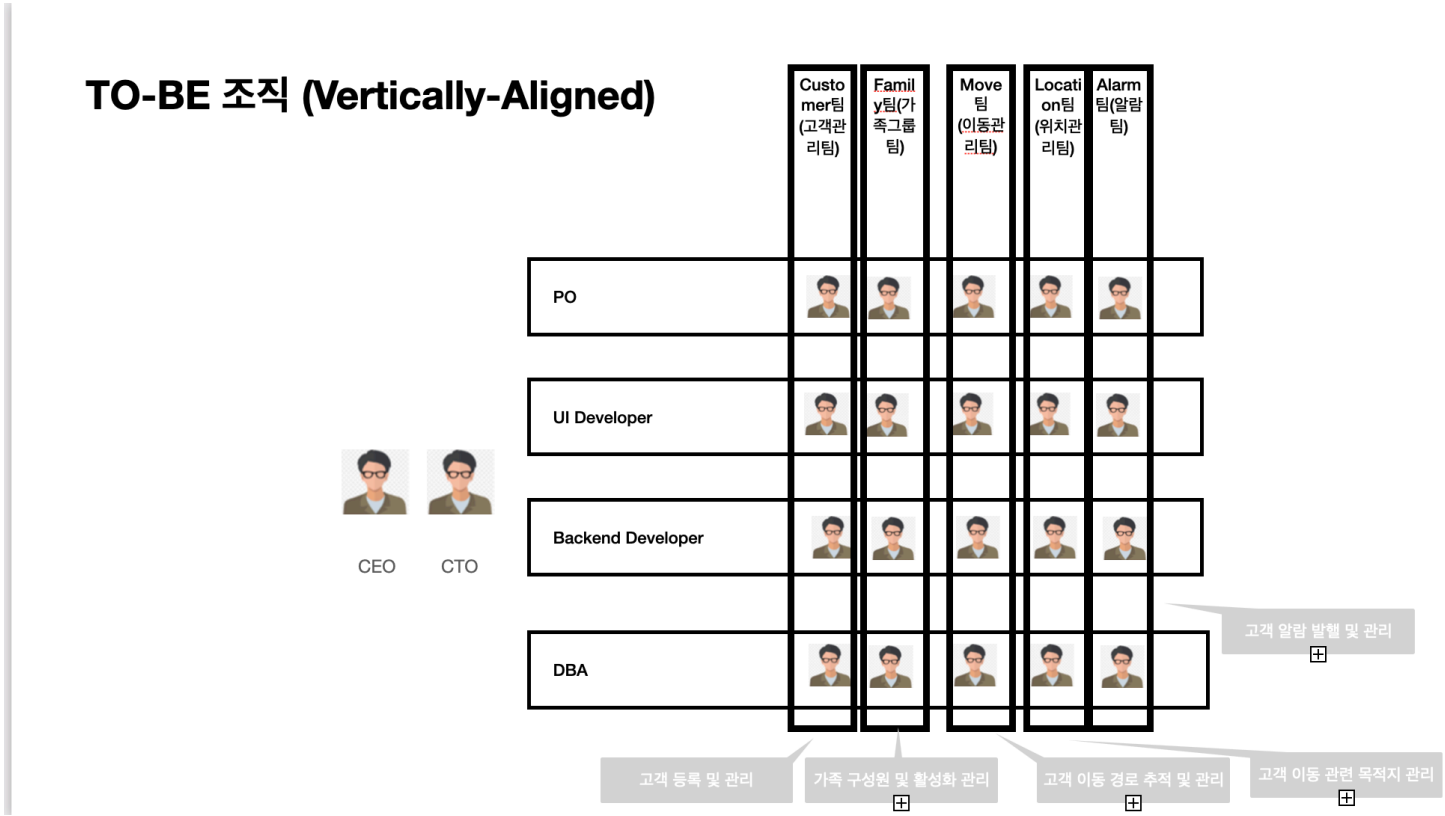
5. 가족이 이동상태를 중간중간 조회한다.
6. 목적지가 바뀌거나/도착 10분전 마다 웹으로 알림을 보낸다.

비기능적 요구사항

- 이동을 취소한 가족구성원을 지속적으로 위치 요청을 하면 안된다. (Sync 호출)
- 모든 가족 구성원에 대한 위치 정보 및 이동 상태 등을 한번에 확인할 수 있어야한다. (CQRS)
- 도착 상태가 바뀔 때마다 웹 알림을 줄 수 있어야 한다. (Event driven)

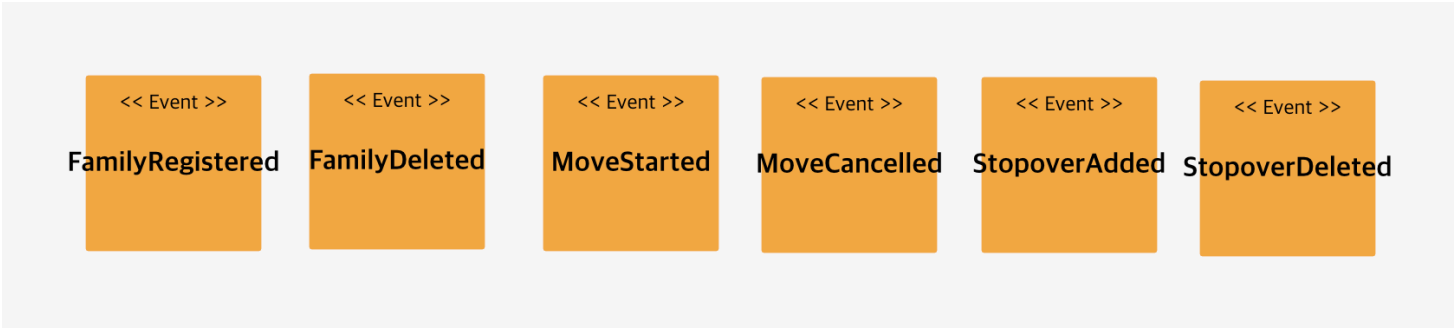
분석 설계



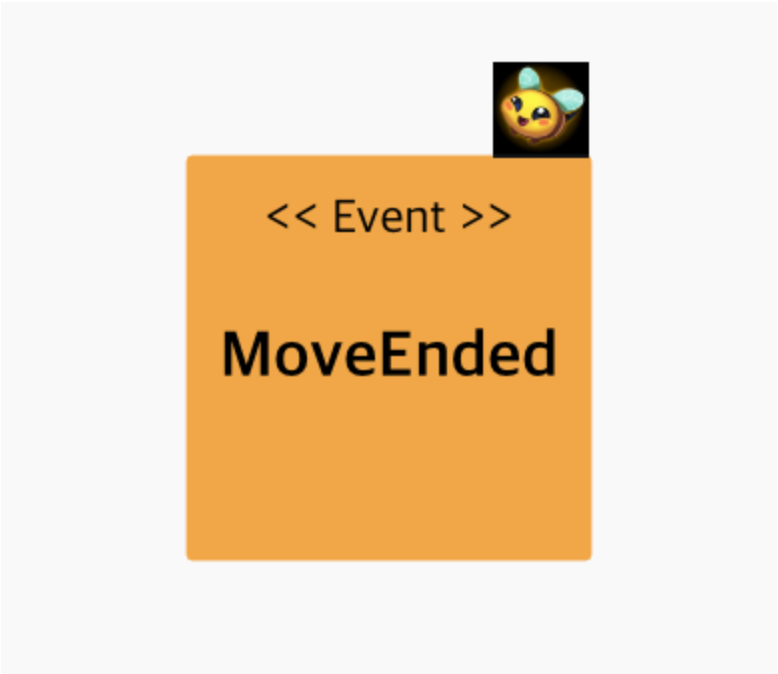


Event Storming 결과

이벤트 도출

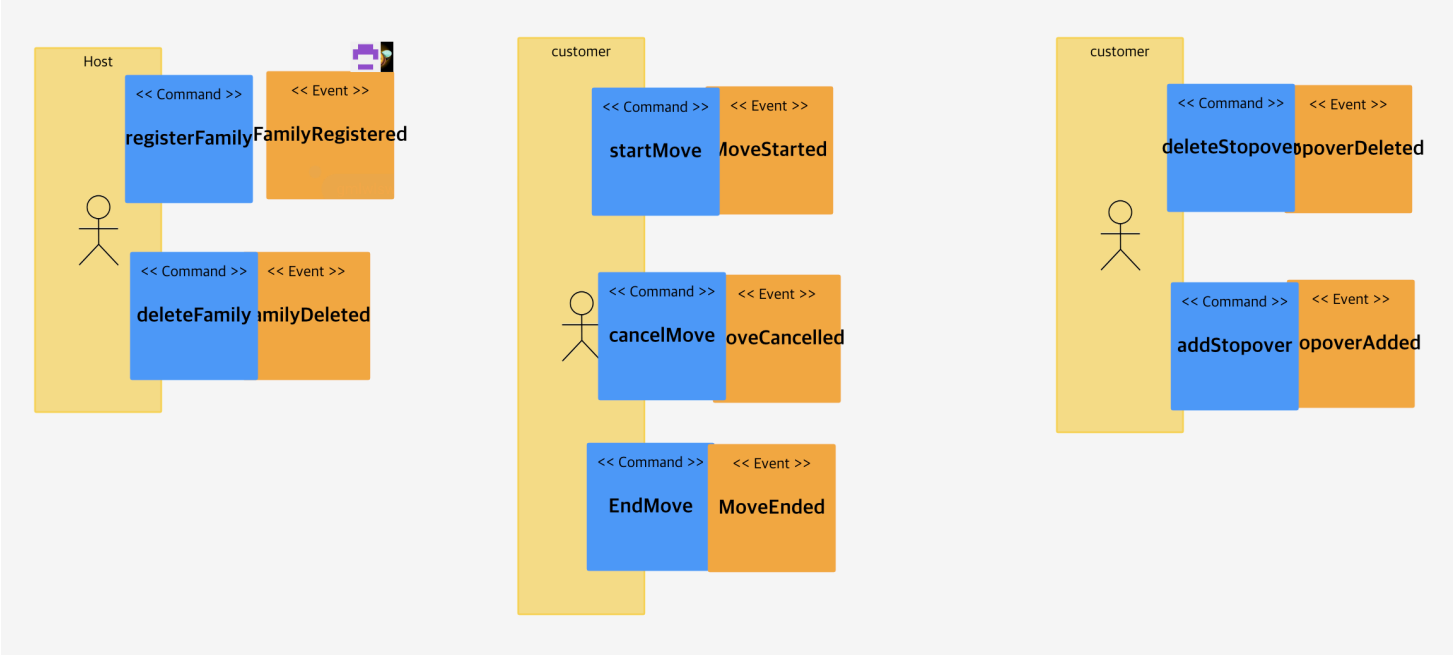


추가 이벤트 도출

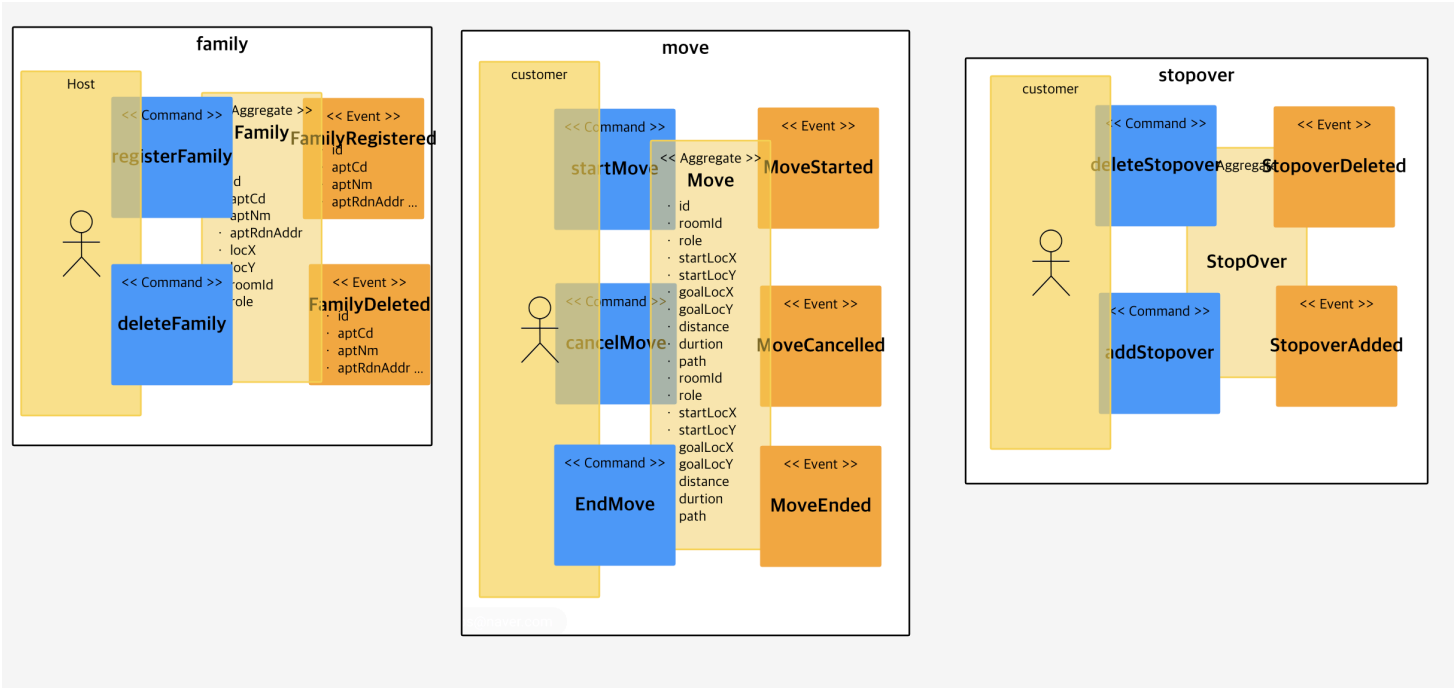


이동이 종료되었을 때 Event를 생성할 필요가 있음

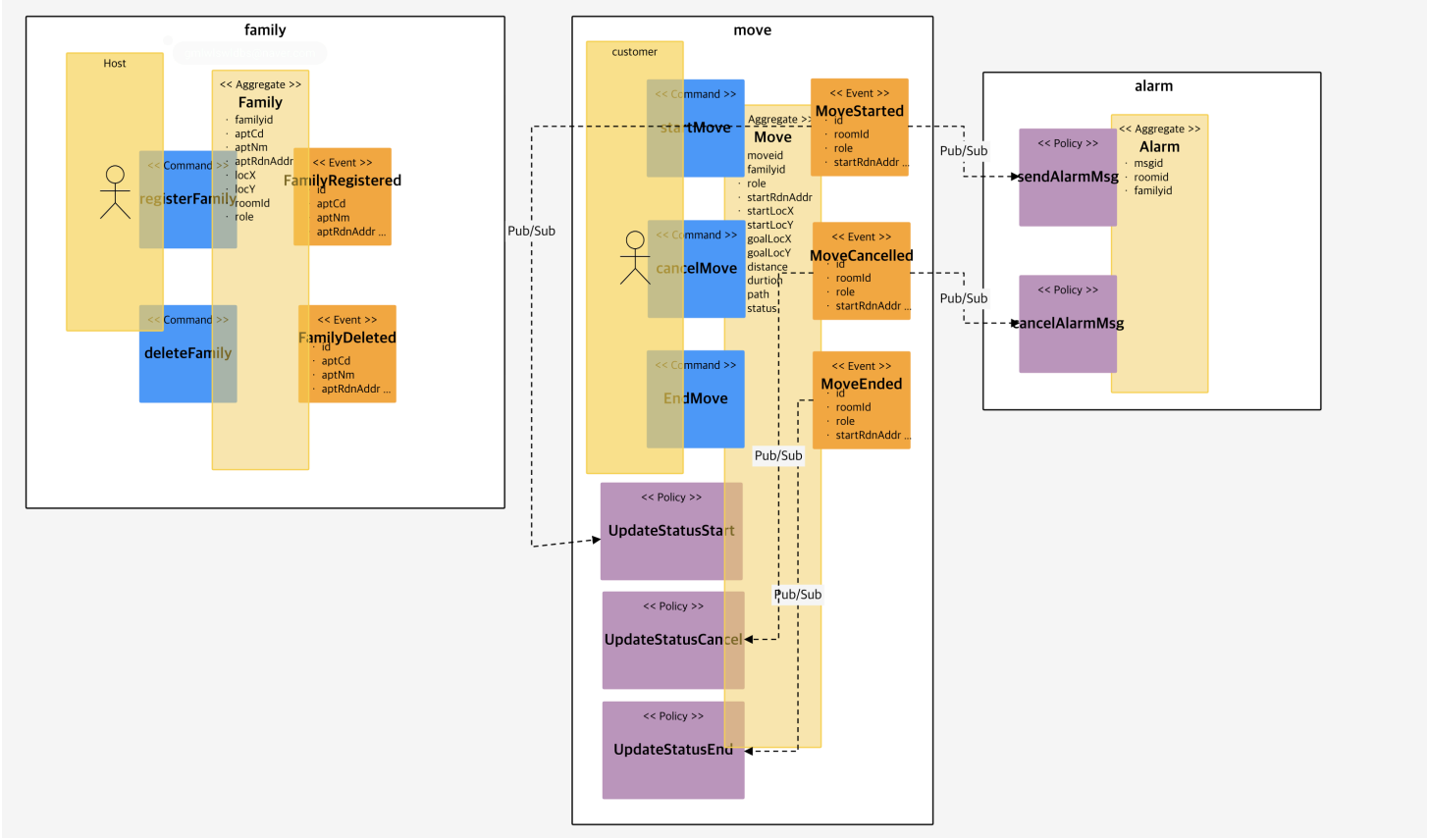
액터, 커맨드 연결



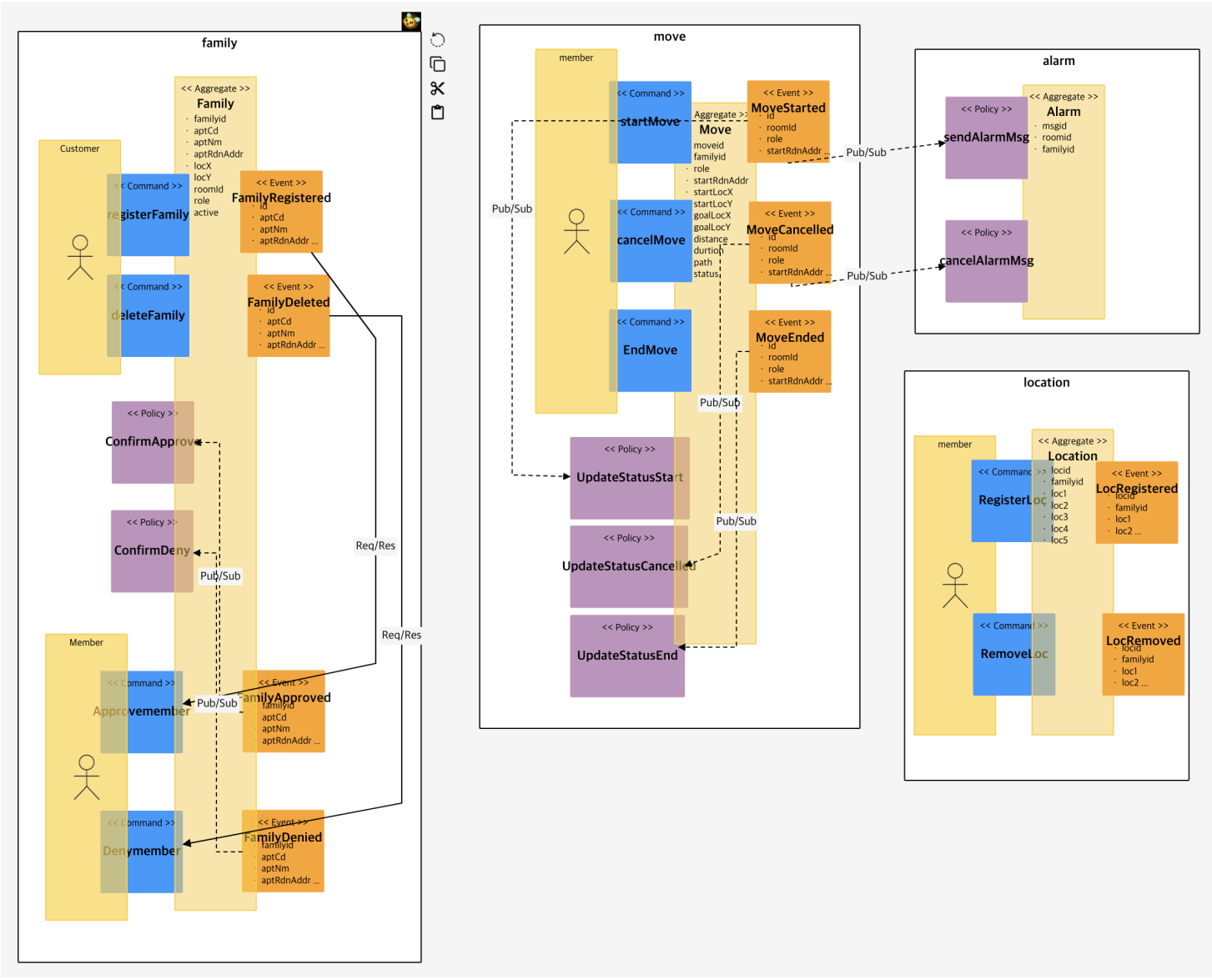
어그리게잇 묶기 및 바운디드 컨텍스트 매핑



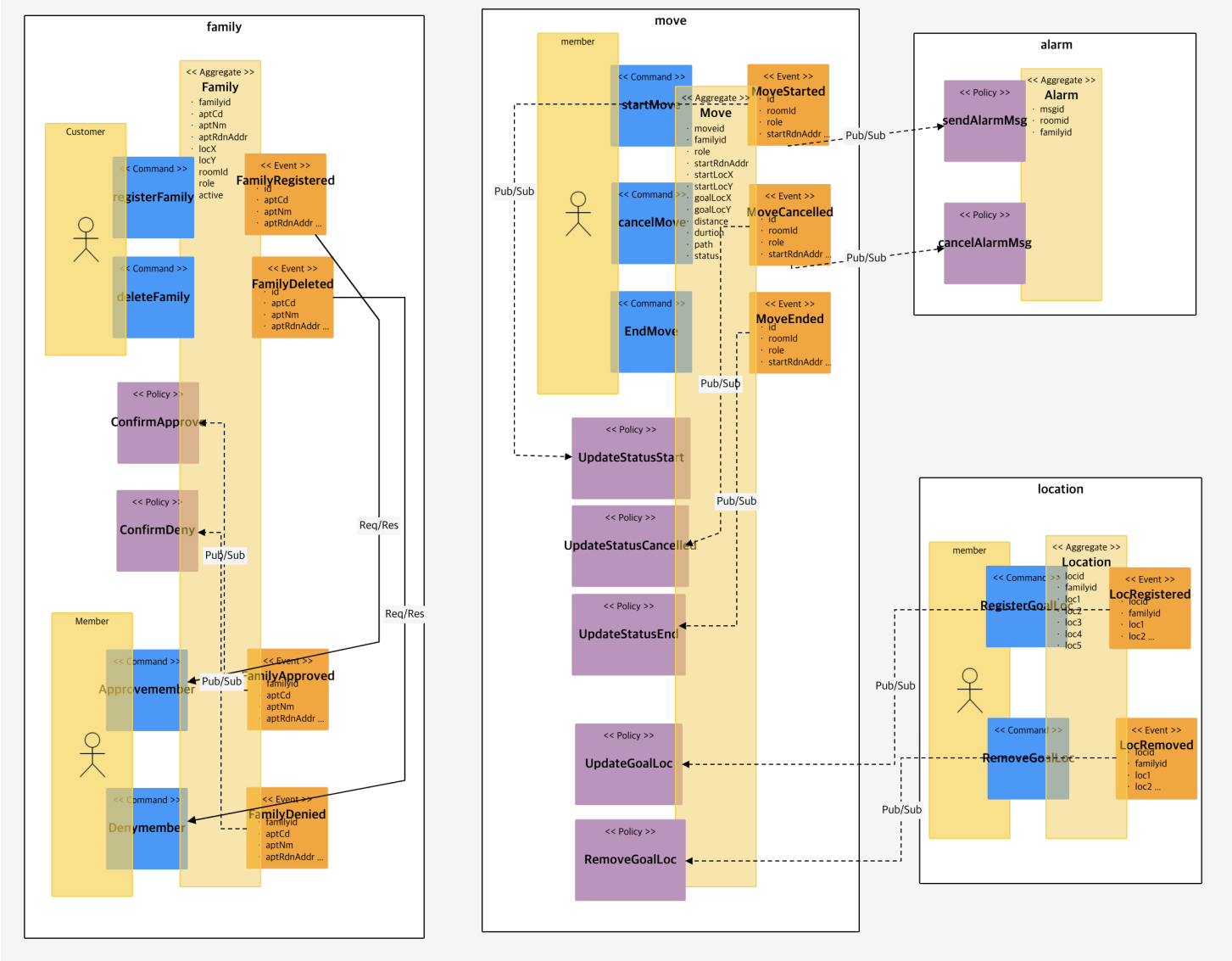
모델 수정 및 비기능 요구사항 검증



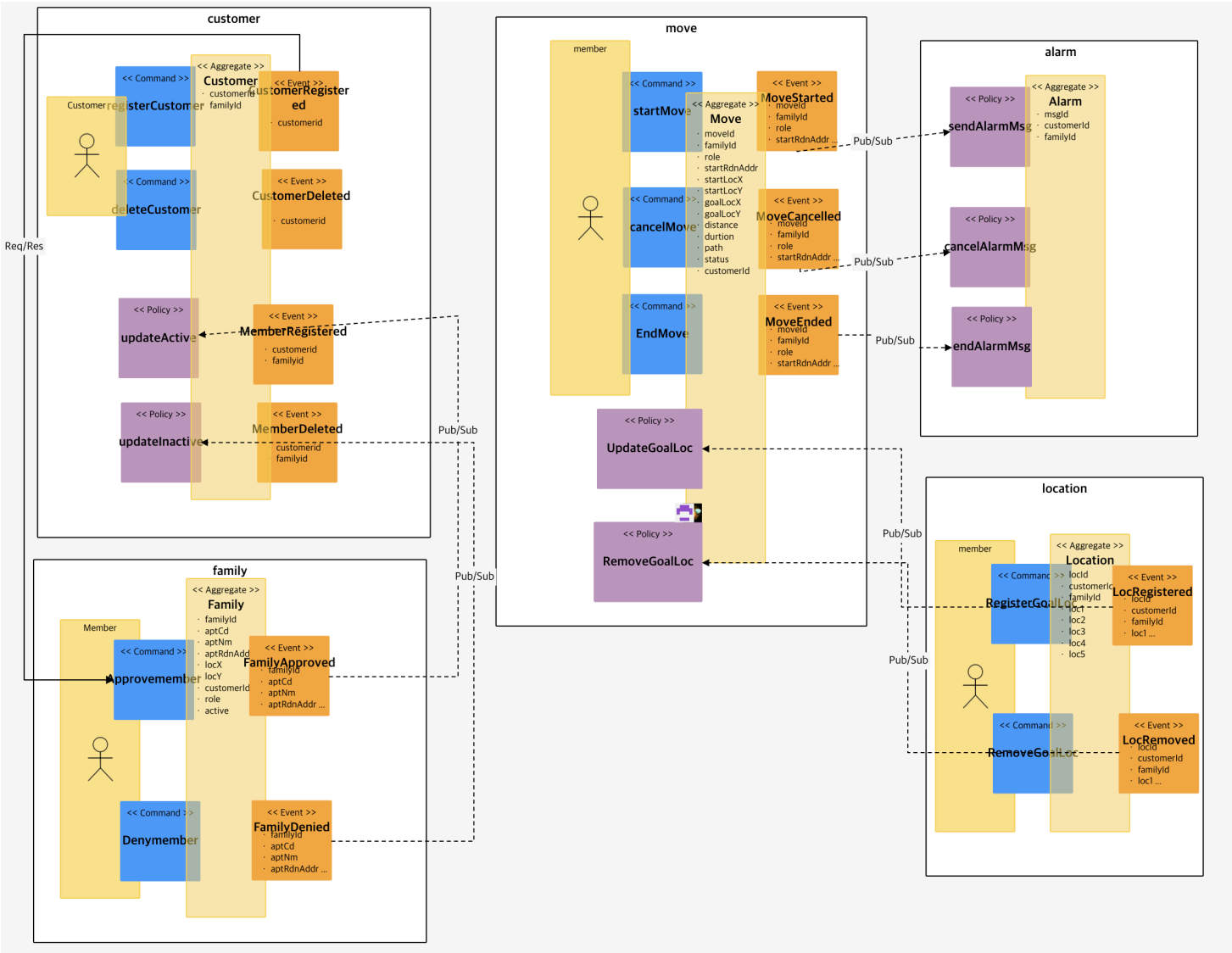
가족구성원 승인여부 및 위치 등록 요구사항 수정



목적지 위치 등록에 따른 move Policy 변경



최종 (불필요한 Policy 제거, customer/family 로직 분기)



사용자와 가족단위를 함께 승인 및 거절 처리하는 것은 어렵다고 판단하여, 컨텍스트를 분리.

회원가입 시 가족을 등록하면 가족 구성원이 승인 및 거절 판별 후에 승인 시 가족이 되는 식으로 변경.

Family가 되기 위해서 Customer가 Register가 된 후 바로 Command로 실행하는 것이 아닌 Policy를 통해 한 번 Event를 받고 동작하도록 수정

Alarm은 알람을 준다는 행위 자체를 한 개의 Policy로 보고 3개의 Command로 나뉘어져 있던 것을 한 개로 통합.

구현

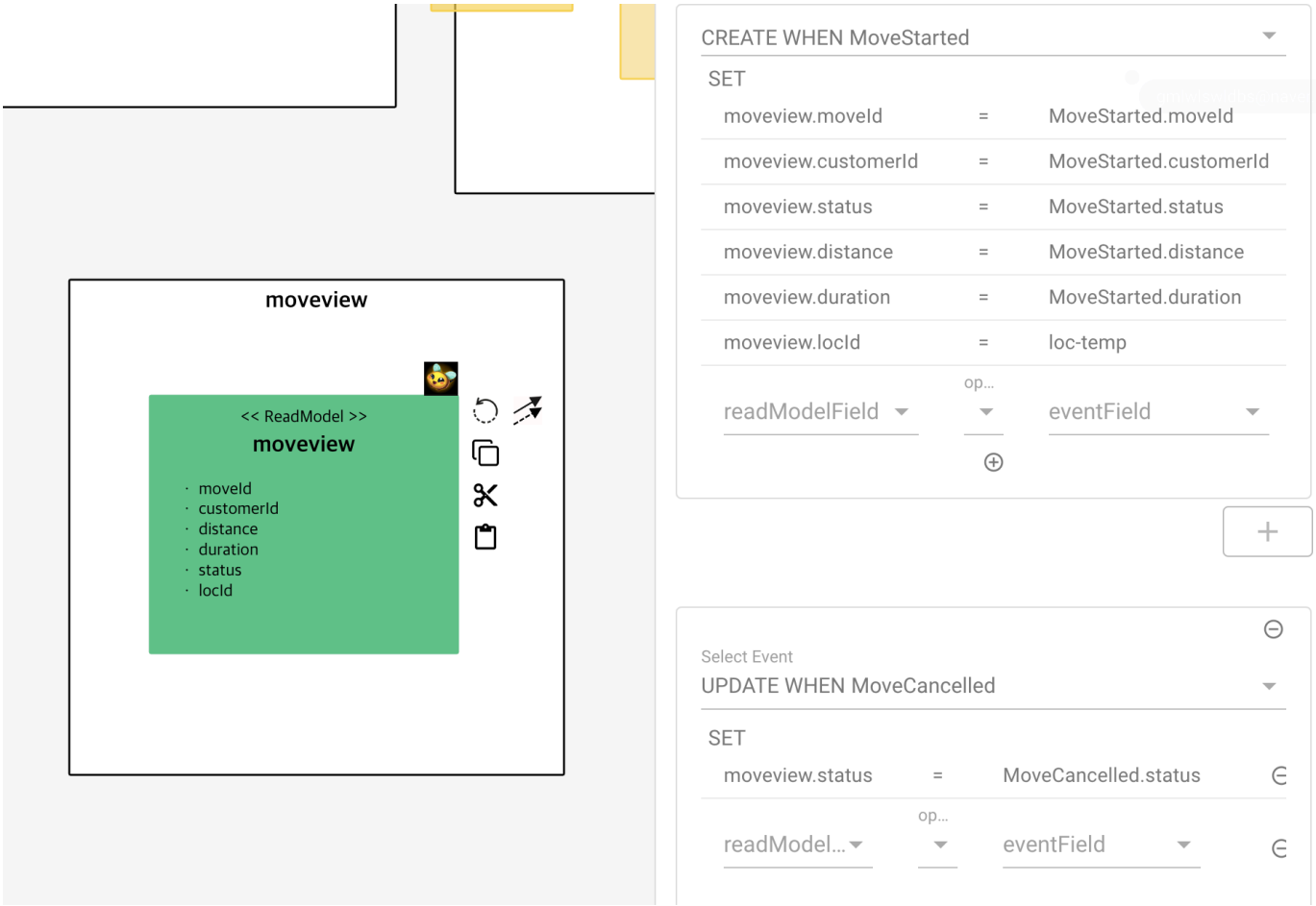
각 BC별로 대변되는 마이크로 서비스들을 스프링부트로 구현. 구현한 각 서비스를 로컬에서 실행하는 방법은 다음과 같다. (사용 포트 넘버는 8081 ~ 808n)

```
mvn spring-boot:run
```

CQRS

이동(Move), 위치(Location) 등 Status에 대하여 고객(Customer)와 가족(Family)가 조회 및 알람을 받을 수 있도록 CQRS로 구현.

- Customer 개별 distace, duration, status Aggregate Status를 moveview를 통해 통합 조회하여 성능 Issue를 사전에 예방할 수 있다.
- 비동기식으로 처리되며, Kafka를 통해 수신/처리 되어 별도 Table에 관리
- CustomerRegister 이벤트 발생 시, Pub/Sub 기반으로 FamilyId에 해당하는 구성원이 수락해야 계정이 Active가 되도록 구성



CQRS 데이터프로젝션을 moveview로 구현하여 API 요청시 customerId에 따른 상태값을 조회할 수 있다.

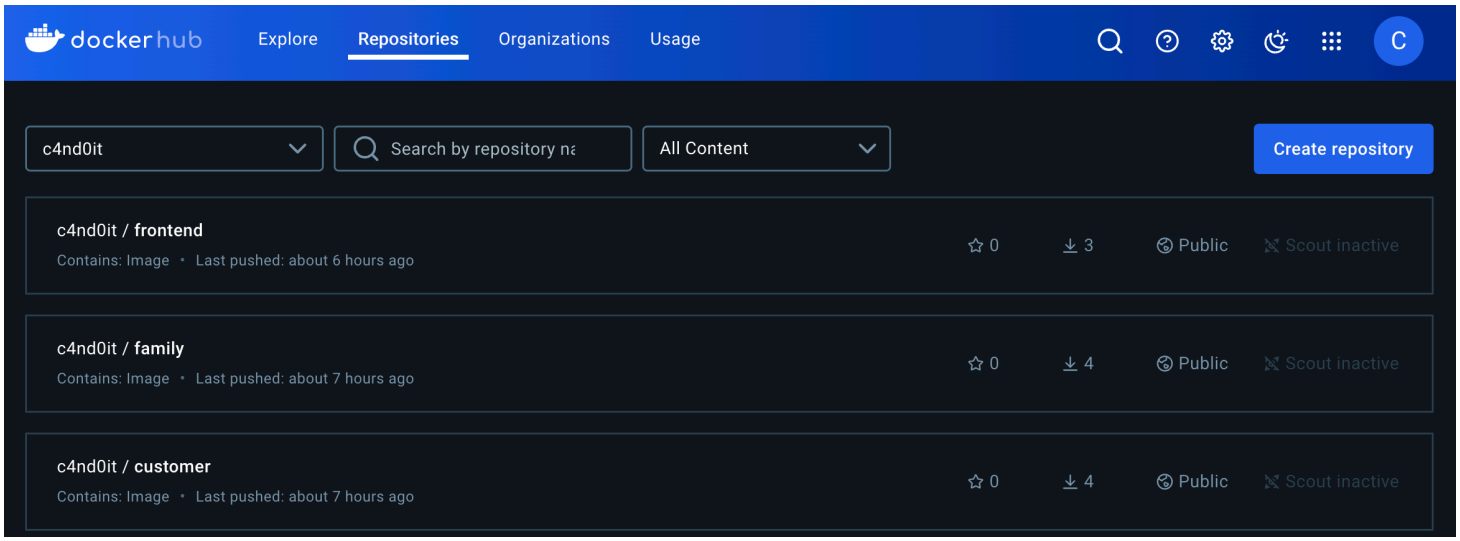
```
{
    "_embedded": {
        "moveviews": [
            {
                "customerId": "guest2",
                "distance": 25220,
                "duration": 1630132,
                "status": "moving",
                "locId": "loc-20241011013733",
                "_links": {
                    "self": {
                        "href": "http://localhost:8087/moveviews/move-20241011013732"
                    },
                    "moveview": {
                        "href": "http://localhost:8087/moveviews/move-20241011013732"
                    }
                }
            },
            {
                "customerId": "guest1",
                "distance": 10328,
                "duration": 1334892,
                "status": "moving",
                "locId": "loc-20241011013746",
                "_links": {
                    "self": {
                        "href": "http://localhost:8087/moveviews/move-20241011013746"
                    },
                    "moveview": {
                        "href": "http://localhost:8087/moveviews/move-20241011013746"
                    }
                }
            },
            {
                "customerId": "guest1",
                "distance": 19217,
                "duration": 1543509,
                "status": "moving",
                "locId": "loc-20241011014036",
                "_links": {
                    "self": {
                        "href": "http://localhost:8087/moveviews/move-20241011014035"
                    },
```

```
            "moveview": {
                "href": "http://localhost:8087/moveviews/move-20241011014035"
            }
        }
    },
    {
        "customerId": "guest1",
        "distance": 19324,
        "duration": 1589128,
        "status": "moving",
        "locId": "loc-20241011014351",
        "_links": {
            "self": {
                "href": "http://localhost:8087/moveviews/move-20241011014350"
            },
            "moveview": {
                "href": "http://localhost:8087/moveviews/move-20241011014350"
            }
        }
    },
    {
        "customerId": "guest1",
        "distance": 10091,
        "duration": 1287967,
        "status": "moving",
        "locId": "loc-20241011014856",
        "_links": {
            "self": {
                "href": "http://localhost:8087/moveviews/move-20241011014856"
            },
            "moveview": {
                "href": "http://localhost:8087/moveviews/move-20241011014856"
            }
        }
    },
    {
        "customerId": "guest1",
        "distance": 25217,
        "duration": 1596921,
        "status": "moving",
        "locId": "loc-20241011015303",
        "_links": {
            "self": {
```

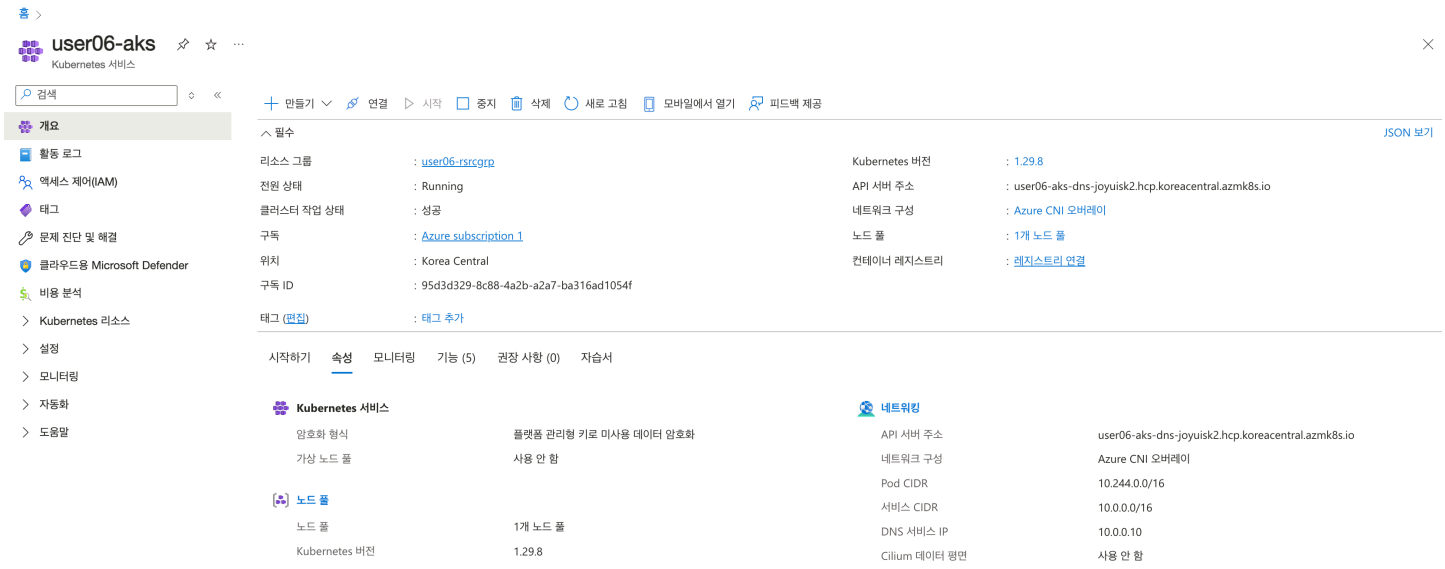
```
                    "href": "http://localhost:8087/moveviews/move-20241011015303"
                },
                "moveview": {
                    "href": "http://localhost:8087/moveviews/move-20241011015303"
                }
            }
        }
    ]
},
"_links": {
    "self": {
        "href": "http://localhost:8087/moveviews"
    },
    "profile": {
        "href": "http://localhost:8087/profile/moveviews"
    }
},
"page": {
    "size": 20,
    "totalElements": 6,
    "totalPages": 1,
    "number": 0
}
}
```

운영

Docker Build 이미지 생성 및 배포(Docker hub)



K8S 배포 설정



Azure Aks에 배포

```
● gitpod /workspace/alpcaproject (main) $ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/customer-5d76d9bd57-bf4bm       1/1     Running   0          17h
pod/family-57bb844b9-2pdlv          1/1     Running   0          17h
pod/frontend-788887757b-8ndkj       1/1     Running   0          15h
pod/gateway-64c55476db-kspf8        1/1     Running   0          11m
pod/location-776786bc85-s8n2n       1/1     Running   0          17h
pod/move-744977df79-q6fqk           1/1     Running   0          18h
pod/moveview-5bccfdb5cb-5gnbc       1/1     Running   0          30s
pod/my-kafka-0                      1/1     Running   0          2d15h

NAME                        TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
service/customer            ClusterIP      10.0.20.46     <none>          8080/TCP                     17h
service/family              ClusterIP      10.0.128.141   <none>          8080/TCP                     17h
service/frontend            ClusterIP      10.0.255.134   <none>          8080/TCP                     15h
service/gateway             LoadBalancer   10.0.215.28    4.230.153.181   8080:31167/TCP               11m
service/kubernetes          ClusterIP      10.0.0.1       <none>          443/TCP                      2d16h
service/location            ClusterIP      10.0.152.121   <none>          8080/TCP                     17h
service/move                ClusterIP      10.0.131.68    <none>          8080/TCP                     18h
service/moveview            ClusterIP      10.0.252.116   <none>          8080/TCP                     30s
service/my-kafka            ClusterIP      10.0.245.1     <none>          9092/TCP                     2d16h
service/my-kafka-headless   ClusterIP      None           <none>          9092/TCP,9094/TCP,9093/TCP   2d16h

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/customer      1/1     1            1           17h
deployment.apps/family        1/1     1            1           17h
deployment.apps/frontend      1/1     1            1           15h
deployment.apps/gateway       1/1     1            1           11m
deployment.apps/location      1/1     1            1           17h
deployment.apps/move          1/1     1            1           18h
deployment.apps/moveview      1/1     1            1           30s

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/customer-5d76d9bd57      1         1         1       17h
replicaset.apps/family-57bb844b9         1         1         1       17h
replicaset.apps/frontend-788887757b      1         1         1       15h
replicaset.apps/gateway-64c55476db       1         1         1       11m
replicaset.apps/location-776786bc85      1         1         1       17h
replicaset.apps/move-744977df79          1         1         1       18h
replicaset.apps/moveview-5bccfdb5cb      1         1         1       30s
```

AKS Pod, service, deployment, replicaset 들이 정상적으로 배포되고 LoadBalancer를 통해 잘 외부 IP에 배포된 모습

API GateWay

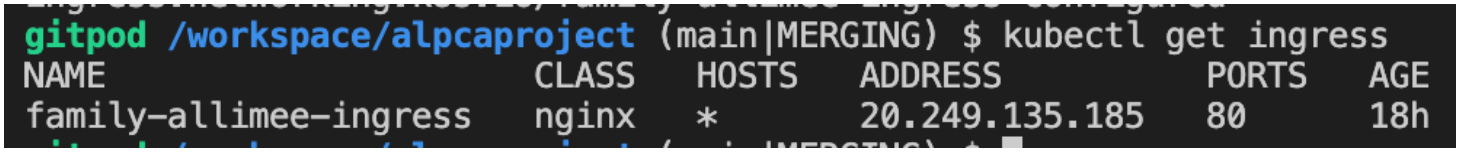
```yaml
server:
  port: 8088

---

spring:
  profiles: default
  cloud:
    gateway:
#<<< API Gateway / Routes
      routes:
        - id: customer
          uri: http://localhost:8082
          predicates:
            - Path=/customers/**,
        - id: move
          uri: http://localhost:8083
          predicates:
            - Path=/moves/**,
        - id: family
          uri: http://localhost:8084
          predicates:
            - Path=/families/**,
        - id: alarm
          uri: http://localhost:8085
          predicates:
            - Path=/alarms/**,
        - id: location
          uri: http://localhost:8086
          predicates:
            - Path=/locations/**,
        - id: frontend
          uri: http://localhost:8080
          predicates:
            - Path=/**
#>>> API Gateway / Routes
      globalcors:
        corsConfigurations:
          '[/**]':
            allowedOrigins:
              - "*"
```

```
            allowedMethods:
              - "*"
            allowedHeaders:
              - "*"
            allowCredentials: true


---

spring:
  profiles: docker
  cloud:
    gateway:
      routes:
        - id: customer
          uri: http://customer:8080
          predicates:
            - Path=/customers/**,
        - id: move
          uri: http://move:8080
          predicates:
            - Path=/moves/**,
        - id: family
          uri: http://family:8080
          predicates:
            - Path=/families/**,
        - id: alarm
          uri: http://alarm:8080
          predicates:
            - Path=/alarms/**,
        - id: location
          uri: http://location:8080
          predicates:
            - Path=/locations/**,
        - id: frontend
          uri: http://frontend:8080
          predicates:
            - Path=/**
      globalcors:
        corsConfigurations:
          '[/**]':
            allowedOrigins:
              - "*"
```

```
            allowedMethods:
              - "*"
            allowedHeaders:
              - "*"
            allowCredentials: true

  server:
    port: 8080
```

Ingress 설정

Ingress를 사용하여 GateWay 포워딩 진행



```yaml
apiVersion: networking.k8s.io/v1
kind: "Ingress"
metadata:
  name: "family-allimee-ingress"
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  ingressClassName: nginx
  rules:
    - host: ""
      http:
        paths:
          - path: /*
            pathType: Prefix
            backend:
              service:
                name: frontend
                port:
                  number: 8080
          - path: /customers
            pathType: Prefix
            backend:
              service:
                name: customer
                port:
                  number: 8080
          - path: /locations
            pathType: Prefix
            backend:
              service:
                name: location
                port:
                  number: 8080
          - path: /alarms
            pathType: Prefix
            backend:
              service:
                name: alarm
                port:
                  number: 8080
          - path: /moves
            pathType: Prefix
```

```
      backend:
        service:
          name: move
          port:
            number: 8080
    - path: /families
      pathType: Prefix
      backend:
        service:
          name: family
          port:
            number: 8080
    - path: /moveviews
      pathType: Prefix
      backend:
        service:
          name: moveview
          port:
            number: 8080
```

Secret

API Key는 따로 관리하기 위해 yaml 파일로 관리 후 변수로 로드

secret.yaml > {} data

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secret-config
  namespace: default
data:
  NAVER_MAPS_CLIENT_ID: [redacted]
  NAVER_MAPS_CLIENT_TOKEN: [redacted]
```

```yaml
env:
- name: naver-maps-client-id
  valueFrom:
    secretKeyRef:
      name: secret-config
      key: naver-maps-client-id
- name: naver-maps-client-token
  valueFrom:
    secretKeyRef:
      name: secret-config
      key: naver-maps-client-token
```