시스템 아키텍처 다이어그램 모음

목차

---

1. Orchestrator 모듈 뷰

목적: Orchestrator의 컴포넌트 구성과 의존성 표현
버전: v9
최종 업데이트: 2025-10-20

plantuml

```
@startuml
title Orchestrator Module View — Composition for Service Lifecycles (v9)
top to bottom direction

package "config" {
  class "ConfigLoader" as CFG {
    + get_settings(userId: str, tenant: str): OrchestrationSettings
  }
}

package "tracker" {
  class "TaskTracker" as TT {
    + get_history(sessionId: str, userId: str): HistorySummary
    + is_current_group_complete(planId: str): bool
    + get_aggregated_results_for_group(planId: str): AggregatedGroupResults
    + persist_plan(plan: Plan): void
    + persist_plan_update(update: PlanUpdate): void
    + append_step_result(result: StepResult): void
    + finalize_conversation(final: FinalSummary): void
  }
}

package "listener" {
  class "ResultListener" as RL {
    - taskTracker: TaskTracker
    - orchestrator: Orchestrator
    --
    + on_result_received(result: StepResult): void
    + start_consuming(): void
  }
}

package "orchestrator" {
  class "Orchestrator" as ORC {
    - taskTracker: TaskTracker
    - documentManager: DocumentManager
    - planner: Planner
    - taskDispatcher: TaskDispatcher
    --
    + run(state: State): State
  }
}

package "state" {
```

```
  interface IRunnable {
    + invoke(state: State): State
  }

  class "DocumentManager" as DM {
    + invoke(state: State): State
    --
    ..Supported states..
    RAG_BUILD_CONTEXT
    --
    - request_user_data(text: str): RagUserData
  }

  class "Planner" as PLN {
    + invoke(state: State): State
    --
    ..Supported states..
    PLAN_OR_DECIDE
    --
    - plan(requestText: str, context: ContextBundle, settings: OrchestrationSettings): Plan
    - decide_next(results: AggregatedGroupResults, planState: PlanState, context: ContextBundle): Decis
  }

  class "TaskDispatcher" as TD {
    + invoke(state: State): State
    --
    ..Supported states..
    DISPATCH
    --
    - publish_step_requested(step: Step, trace: TraceContext): void
  }
}

' Implements
DM  ..|> IRunnable
PLN ..|> IRunnable
TD  ..|> IRunnable

' Orchestrator composition
ORC *-- DM
ORC *-- PLN
ORC *-- TD

' Dependencies
```

```
ORC ..> TT : uses
ORC ..> CFG : get_settings()

' ResultListener relationships
RL ..> TT : append_step_result()
RL ..> ORC : run()

' layout nudge
CFG -[hidden]down- TT
TT -[hidden]down- RL
RL -[hidden]down- ORC
ORC -[hidden]down- DM

@enduml
```

주요 구성요소:

- **ConfigLoader**: 사용자/테넌트별 설정 로드 (최초 1회)
- **TaskTracker**: 작업 상태 및 히스토리 관리 유틸리티
- **ResultListener**: MQ로부터 작업 결과 수신 및 Orchestrator 재개
- **Orchestrator**: 전체 워크플로우 조율
- **State 패키지**: DocumentManager, Planner, TaskDispatcher (IRunnable 구현)

---

2. Orchestration Service - 요청부터 플래닝까지

목적: 사용자 요청 수신부터 실행 계획 수립까지의 흐름
최종 업데이트: 2025-10-20

plantuml

```
@startuml
title Orchestration Service (요청 → 플래닝)
autonumber

participant "API Gateway" as GW
participant "Orchestrator" as ORC
participant "ConfigLoader" as CFG
participant "DocumentManager\n(Runnable)" as DM
participant "Planner\n(Runnable)" as PLN
actor "LLM\n(External)" as LLM

== 요청 수신 & 설정 ==
GW -> ORC : User Request(payload, trace_id)
note over GW,ORC
  session_id, user_id, tenant,
  request_text, trace
end note

ORC -> CFG : get_settings(user_id, tenant)
CFG --> ORC : settings(agents, llm, rpa, policies)

== RAG (사용자 요청 + 히스토리 병합) ==
ORC -> DM : invoke(RAG_BUILD_CONTEXT, state{request_text, session_id})
note right of DM
  내부적으로:
  1. TaskTracker에서 히스토리 조회
  2. RAG Service 호출로 context 구축
end note
DM --> ORC : state{context}

== 플래닝 ==
ORC -> PLN : invoke(PLAN_OR_DECIDE, state{request_text, context, settings})

PLN -> PLN : split_and_route_llm(request_text, available_tools)
PLN -> LLM : generate(분해 프롬프트)
LLM --> PLN : units[]

loop for each unit
  PLN -> PLN : select_prompt_template(unit.tool_id)
  PLN -> PLN : render_messages(template, context, unit.text)

  PLN -> LLM : generate(스텝 생성 프롬프트)
  LLM --> PLN : steps[]
```

```
  PLN -> LLM : generate(정규화 프롬프트)
  LLM --> PLN : normalized_steps[]
 end

 PLN -> PLN : assemble_plan(steps, deps)

 alt 플래닝 성공
  PLN --> ORC : state{plan(steps, deps, guards)}
  note right of ORC
   TaskTracker를 통해
   Plan을 영속화
  end note
 else 플래닝 실패
  PLN --> ORC : state{error, reason}
  ORC -> ORC : transition_to(HUMAN_IN_THE_LOOP)
  note over ORC
   사람의 개입이 필요한 상태로 전환
   - 불명확한 요청
   - 사용 가능한 도구 없음
   - 제약 조건 위반 등
  end note
 end

 @enduml
```

주요 단계:

1. **요청 수신 & 설정**: API Gateway → Orchestrator, 설정 로드
2. **RAG**: DocumentManager가 히스토리 조회 및 컨텍스트 구축
3. **플래닝**: Planner가 LLM을 사용해 요청 분해 → 스텝 생성 → 정규화 → Plan 조립

---

3. Orchestration Service - 실행 루프

목적: Plan 실행 및 작업 완료까지의 반복 루프
최종 업데이트: 2025-10-20

plantuml

```
@startuml
title Orchestration Service — Execution Loop
autonumber

participant "API Gateway" as GW
participant "Orchestrator" as ORC
participant "TaskTracker" as TT
participant "Planner\n(Runnable)" as PLN
participant "TaskDispatcher\n(Runnable)" as TD
participant "ResultListener" as RL
participant "Message Queue" as MQ

note over ORC
  이전 단계에서 Plan 생성 완료
  (요청 수신 → 설정 → RAG → 플래닝)
end note

== 실행 루프 (최대 10회) ==
loop 최대 10회 또는 종료 조건 만족 시까지

  group 1. 작업 디스패치
    ORC -> TD : invoke(DISPATCH, state{plan, trace})
    TD -> MQ : publish("task.step.requested")
    note right: 외부 Task Service에서 처리
  end

  group 2. 결과 수집 (비동기)
    MQ -> RL : consume("task.step.succeeded/failed")
    RL -> TT : append_step_result(result)
    RL -> RL : check if group complete

    alt 그룹 완료
      RL -> TT : get_aggregated_results_for_group(plan_id)
      TT --> RL : aggregated_results
      RL -> ORC : on_group_completed(aggregated_results)
    end
  end

  group 3. 다음 행동 결정
    ORC -> PLN : invoke(PLAN_OR_DECIDE, state{results, plan_state, context})
    note right
      Planner가 판단:
      - 작업 완료? → final
      - 다음 단계 필요? → nextSteps
```

```
    - 불명확/실패? → HITL
  end note
  PLN --> ORC : decision{type, payload}

  alt nextSteps
    ORC -> TT : persist_plan_update(decision.steps)
    note over ORC: 다음 루프 계속

  else final
    ORC -> TT : finalize_conversation(final_payload)
    ORC --> GW : 200 OK (final result)
    break

  else HITL (count >= 3이면 실패 처리)
    ORC -> TT : finalize_conversation(HITL_payload or FAILED)
    ORC --> GW : 200 OK (HITL required or failed)
    break
  end
 end

end

@enduml
```

주요 단계:

1. **작업 디스패치**: TaskDispatcher가 MQ에 작업 요청 발행
2. **결과 수집**: ResultListener가 MQ로부터 결과 수신 및 그룹 완료 체크
3. **다음 행동 결정**: Planner가 완료/계속/HITL 판단

예상 처리량 (4,000명, 10% 접속률, 15초 주기):

- 초당 약 800건
- 분당 약 48,000건
- 월간 약 7천만 건

---

4. RAG Service - 클래스 다이어그램

목적: RAG Service의 클래스 구조 및 관계
최종 업데이트: 2025-10-20

plantuml

```
@startuml
title RAG Service - Class Diagram

' Core Components
class Retriever {
  + retrieve(apiKey: string, question: string, filters: Map, topK: int): List<SearchHit>
  + storeDocuments(apiKey: string, files: List<File>, meta: Map): UpsertResult
  --
  - validateApiKey(apiKey: string): Scope
  - toQueryVector(q: string): Vector
}

class DocumentParser {
  + parse(file: File, meta: Map): List<Chunk>
  --
  - normalize(text: string): string
  - chunk(text: string): List<Chunk>
}

' LangChain 통합
class DocumentParser {
  <<LangChain BaseDocumentLoader>>
}

class TextSplitter {
  <<LangChain TextSplitter>>
}

class EmbeddingClient {
  <<LangChain Embeddings>>
  + embed(texts: List<string>): List<Vector>
  --
  - modelName: string
  - dim: int
}

class VectorDBClient {
  + search(scope: Scope, vector: Vector, filters: Map, topK: int): List<SearchHit>
  + upsert(scope: Scope, vectors: List<Vector>, meta: Map): UpsertResult
  --
  - indexName: string
  - namespacePrefix: string
}
```

```
' Value Objects
class Scope <<value object>> {
  + tenantId: string
  + userId: string
  + namespace(): string
}

' LangGraph 통합
class ResultListener {
  + onTaskComplete(taskResult: TaskResult): void
  --
  - triggerNextNode(): void
}

' Relationships
Retriever --> DocumentParser : uses
Retriever --> EmbeddingClient : uses
Retriever --> VectorDBClient : uses
Retriever ..> Scope : resolves from API key
VectorDBClient ..> Scope : uses for DLS
DocumentParser --> TextSplitter : uses

note right of Retriever
  API Key 검증 → Scope 생성
  Scope는 DLS에서 tenant_id, user_id 필터링에 사용
end note

note right of VectorDBClient
  OpenSearch Document Level Security:
  JWT의 tenant_id, user_id로 문서 접근 제어
end note

@enduml
```

주요 특징:

- **LangChain 통합**: DocumentParser, TextSplitter, EmbeddingClient가 LangChain 베이스 클래스 상속
- **Document Level Security**: Scope 객체로 테넌트/사용자별 격리
- **LangGraph 통합**: ResultListener가 MQ로부터 완료 메시지 수신 후 워크플로우 진행

---

5. RAG Service - 문서 검색 시퀀스

목적: 사용자 질문에 대한 문서 검색 흐름
최종 업데이트: 2025-10-20

plantuml

```
@startuml
title RAG Service - 문서 검색
autonumber

participant "Orchestration Service" as ORCH
participant "Retriever" as RETR
participant "EmbeddingClient" as EMB
participant "VectorDBClient" as VDB
actor "Embedding Model\n(External)" as EMBMODEL
actor "Vector DB\n(External)" as VECTORDB

ORCH -> RETR : retrieve(apiKey, question, filters, topK)

' API Key 기반 사용자 인증/스코핑
RETR -> RETR : validateApiKey(apiKey)\nresolve user/tenant scope
alt Invalid or unauthorized
  RETR --> ORCH : 401 Unauthorized
end

' Query 임베딩
RETR -> EMB : embed(question)
EMB -> EMBMODEL : API call (text → vector)
EMBMODEL --> EMB : embedding vector
EMB --> RETR : queryVector

' VectorDB에서 Top-K 검색 (DLS 적용)
RETR -> VDB : search(scope=apiKey, vector=queryVector, filters, topK)
VDB -> VECTORDB : query (vector similarity + DLS filter)
note right of VECTORDB
  OpenSearch DLS:
  JWT의 tenant_id, user_id로
  문서 필터링
end note
VECTORDB --> VDB : search results
VDB --> RETR : hits[topK] (docId, chunk, score, meta)

' 검색 결과 반환
RETR --> ORCH : 200 OK {hits}
@enduml
```

보안 특징:

- API Key 검증으로 Scope(tenant_id, user_id) 추출
- OpenSearch DLS로 문서 레벨 접근 제어

- 3계층 보안: Ingress JWT → 마이크로서비스 검증 → OpenSearch DLS

---

6. RAG Service - 문서 업로드 시퀀스

목적: 사용자 문서 업로드 및 벡터화 흐름
최종 업데이트: 2025-10-20

plantuml

```
@startuml
title RAG Service - 문서 업로드
autonumber

participant "Orchestration Service" as ORCH
participant "Retriever" as RETR
participant "DocumentParser" as PARSER
participant "EmbeddingClient" as EMB
participant "VectorDBClient" as VDB
actor "Embedding Model\n(External)" as EMBMODEL
actor "Vector DB\n(External)" as VECTORDB

ORCH -> RETR : storeDocuments(apiKey, files[], meta)

' API Key 검증
RETR -> RETR : validateApiKey(apiKey)\nresolve user/tenant scope
alt Invalid or unauthorized
  RETR --> ORCH : 401 Unauthorized
end

loop for each file
  ' 문서 파싱 및 청킹
  RETR -> PARSER : parse(file, meta)
  PARSER -> PARSER : normalize(text)
  PARSER -> PARSER : chunk(text)
  PARSER --> RETR : chunks[]

  ' 청크 임베딩
  RETR -> EMB : embed(chunks[].text)
  EMB -> EMBMODEL : API call (texts[] → vectors[])
  EMBMODEL --> EMB : embedding vectors[]
  EMB --> RETR : vectors[]

  ' VectorDB 저장 (DLS 메타데이터 포함)
  RETR -> VDB : upsert(scope, vectors[], meta{tenant_id, user_id})
  VDB -> VECTORDB : insert (with DLS metadata)
  note right of VECTORDB
    문서에 tenant_id, user_id 저장
    이후 검색 시 DLS 필터링에 사용
  end note
  VECTORDB --> VDB : success
  VDB --> RETR : UpsertResult
end
```

```
RETR --> ORCH : 200 OK {documentIds[], count}
@enduml
```

처리 단계:

1. **파싱**: DocumentParser가 파일을 청크로 분할
2. **임베딩**: EmbeddingClient가 청크를 벡터로 변환
3. **저장**: VectorDBClient가 tenant_id, user_id 메타데이터와 함께 저장

시스템 스펙 요약

Orchestration Service

- **아키텍처**: LangGraph 기반 상태 머신
- **프레임워크 선택 이유**:
  - 기능 정확도 최상 (9/10)
  - 체크포인팅으로 장애 복구
  - 멀티 에이전트 협업에 최적
- **예상 처리량**: 월 7천만 건 (4,000명, 10% 접속률 기준)

RAG Service

- **벡터 DB**: OpenSearch + Document Level Security
- **임베딩 모델**: 외부 API 호출
- **보안**: JWT 기반 3계층 (Ingress → 서비스 → DLS)
- **테넌트 격리**: 테넌트별 인덱스 + DLS 조합

Message Queue

- **선택**: RabbitMQ (K8s 외부 배치)
- **클러스터**: 3-5 노드
- **처리 목표**: p95 0.5초, p99 1초
- **확장 전략**: Consumer Auto-scaling (HPA)

인프라

- **배포**: Kubernetes
- **스케일링**: HPA (CPU 70% 기준) + Cluster Autoscaler
- **모니터링**: 큐 깊이, 처리 시간, Consumer 수

변경 이력

날짜	다이어그램	주요 변경사항
2025-10-20	Orchestrator 모듈 뷰	ResultListener 추가, TaskTracker 독립 패키지화
2025-10-20	실행 루프	Event-driven 방식 적용, HITL 로직 간소화
2025-10-20	RAG 클래스	LangChain 통합, LangGraph ResultListener 추가
2025-10-20	RAG 시퀀스	Actor 분리 (Embedding Model, Vector DB)

작성일: 2025-10-22
작성자: Architecture Team
문서 버전: 1.0