Couchbase

Couchbase Capella Workshop
> On Mobile/Device AI

2시 5분에 시작하겠습니다.

2024.11.27

손 광락, Solutions Engineer

Capella Webinar

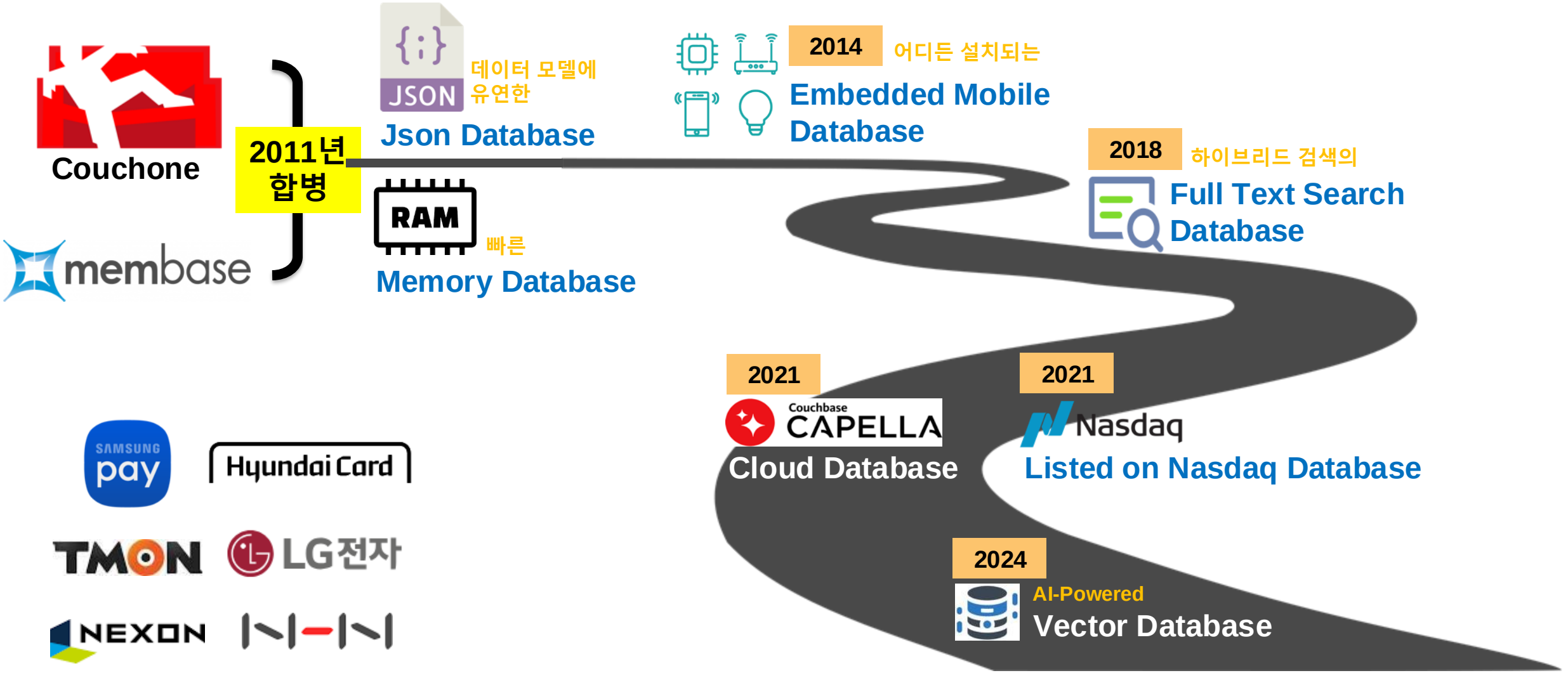
구분	일자	Webinar 주제
시리즈 1	2024-09-25	벡터 검색을 활용한 AI Powered 어플리케이션 구축
시리즈 2	2023-10-30	벡터 검색을 활용한 GenAI(LLM/RAG) 어플리케이션 구축
시리즈 3	**2024-11-27**	**벡터 검색을 활용한 Mobile On-Device AI 어플리케이션 구축**

Agenda

- 14:00 – 14:10 등록 확인 & 실습 환경 확인
- 14:10 – 14:35 카우치베이스 소개
- 14:35 – 15:00 On-Mobile/Device AI 소개
- 15:00 – 15:10 Break
- 15:10 – 15:25 Couchbase SyncGateway 구성 실습
- 15:25 – 15:50 On-Mobile/Device AI 실습/데모
- 15:50 – 16:00 Q & A

Couchbase
Introduction & Architecture

1 개요 : Couchbase 회사 소개



1 개요 : Couchbase 도입 사례

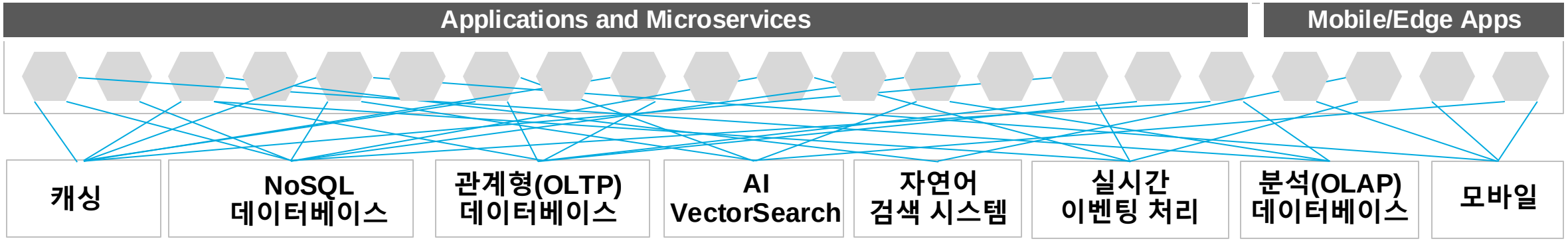
	LinkedIn	TESCO	amadeus	COMCAST	UNITED
Application	캐싱 & 싱글뷰를 위한 세션 스토어	리얼 타임 프라이싱, 제품 캐탈로그, 재고관리	비행편 가용성, 예약, 가격분석등	Customer 360 싱글뷰, ‘Unified notes’App지원	리얼타임 승무원 분석, 일정 및 리소스 관리
Performance	**2백만+** 읽기/초당 **1천만** 쿼리/초당	**1천만+** SKUs **3만5천** 요청/초당.	**8백만** Ops / 초당 **<2.5ms** 반응시간	**2.1억개** 다큐먼트 **10만** 사용자	**4.1만** 종업원 **1.5억+** 이용객

Walmart, cars.com, Fanatics, Office Depot OfficeMax, Staples, Tesco, Best Buy, Marriott, amadeus, Sabre, Orbitz, tripadvisor, Hilton, FICO, Revolut, PayPal, Wells Fargo, Visa, Pfizer, everyday health, BD, Cochlear, Monsenso, Comcast, KDDI, at&t, Viber, Telefónica, sky, Thomson Reuters, Fandango, Gannett, Disney, nielsen, EA, betfair, zynga, Blizzard, Playtika, SGN, verizon, Sierra Wireless, Chamberlain, GE

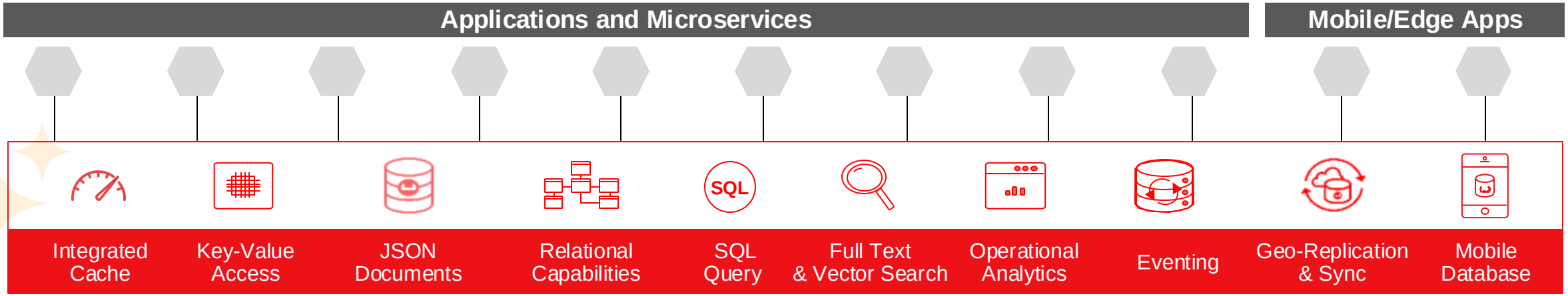
3 of the Top 10 eCommerce Companies
3 of the Top 3 GDS Companies
3 of the Top 3 Credit Reporting Companies
3 Fortune 500 Healthcare Companies

6 of the Top 10 Broadcast Companies
6 of the Top 10 Online Casino Gaming Companies
2 of the Top 2 IoT Platforms

1 개요 : 하나의 App을 만들기 위해 다양한 데이터 관리 솔루션이 필요.

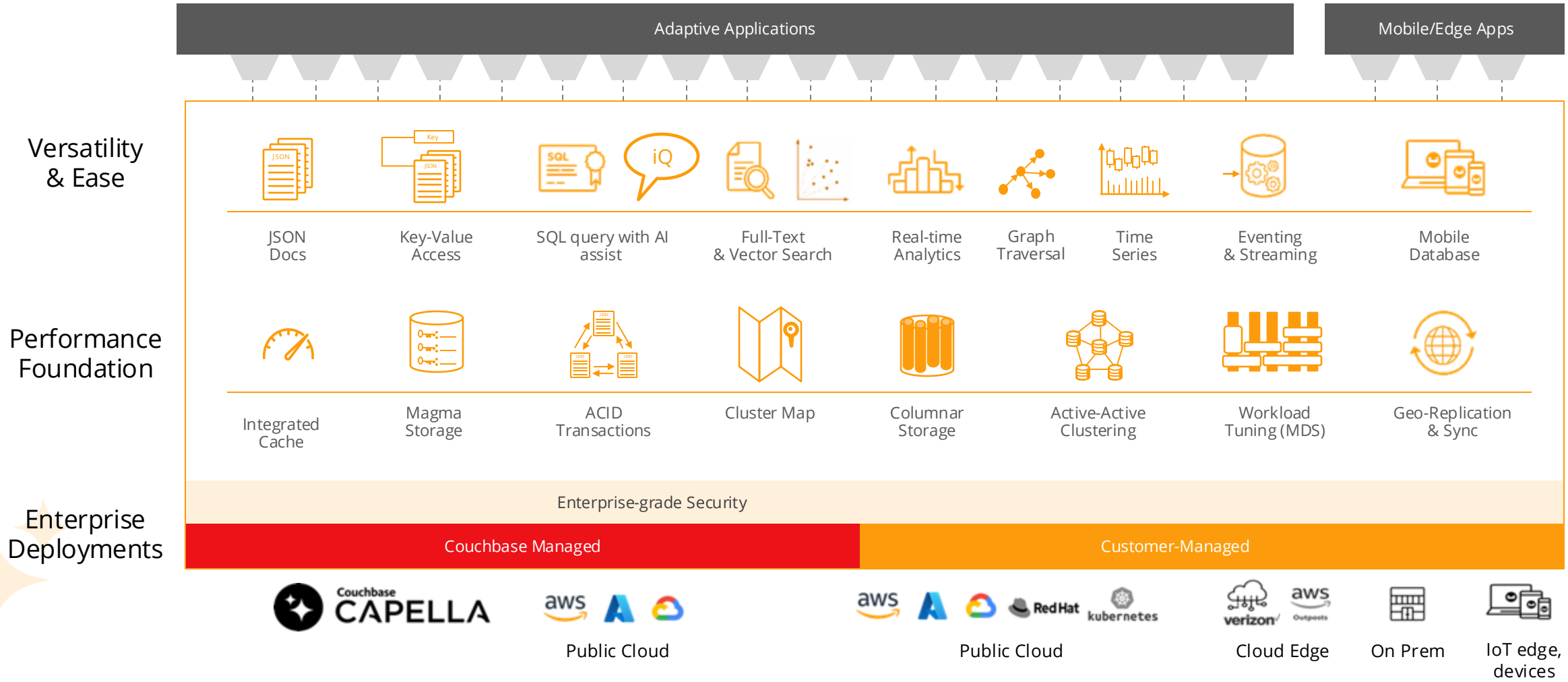


하나의 데이터 플랫폼으로는 할 수 없나?

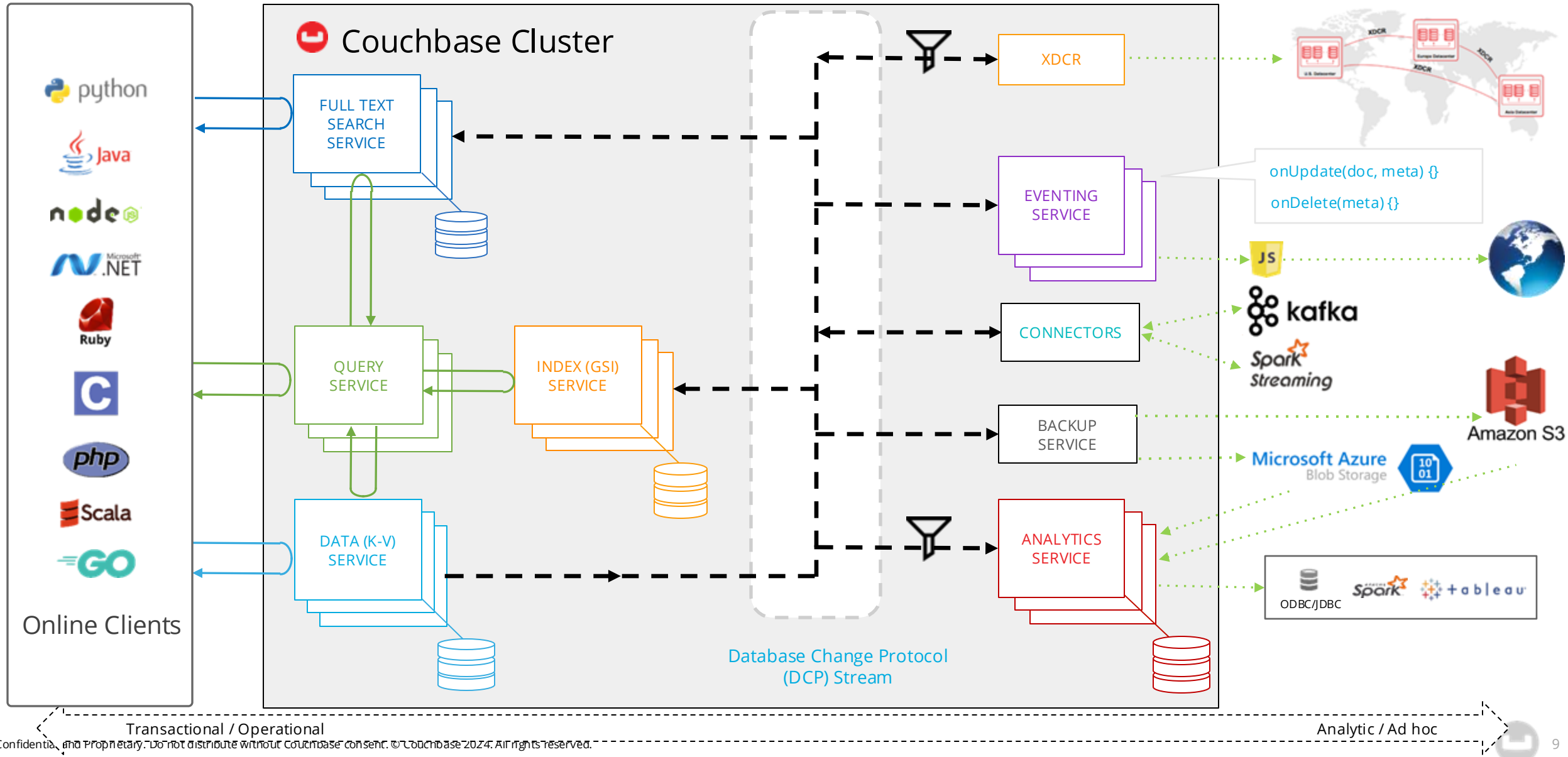


1 개요 : Enterprise 데이터 플랫폼

JSON 도큐먼트 DB, Key-Value 캐시, 표준 SQL, 텍스트 검색, 실시간 분석, 시계열 처리, 고가용성, 재해복구, 모바일 DB 지원하며 AI-Powered 어플리케이션을 위한 데이터 플랫폼입니다.



2 아키텍처 : 메모리 기반 Micro Service 아키텍처

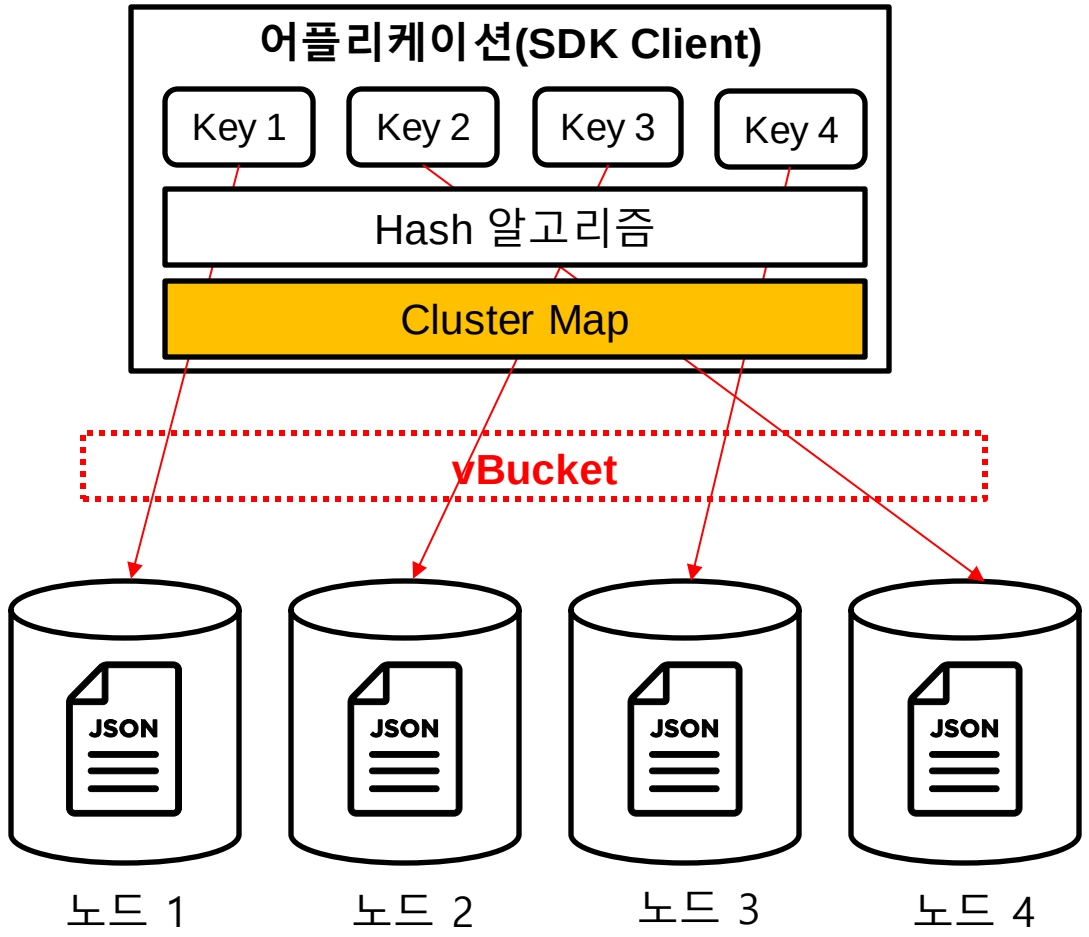


2 아키텍처 : 분산 병렬, Master Node-less 아키텍처

데이터를 다수의 노드에 Key 기반 자동 분산 저장하며, 별도의 마스트 노드없이 모든 노드에서 병렬 처리를 수행함.

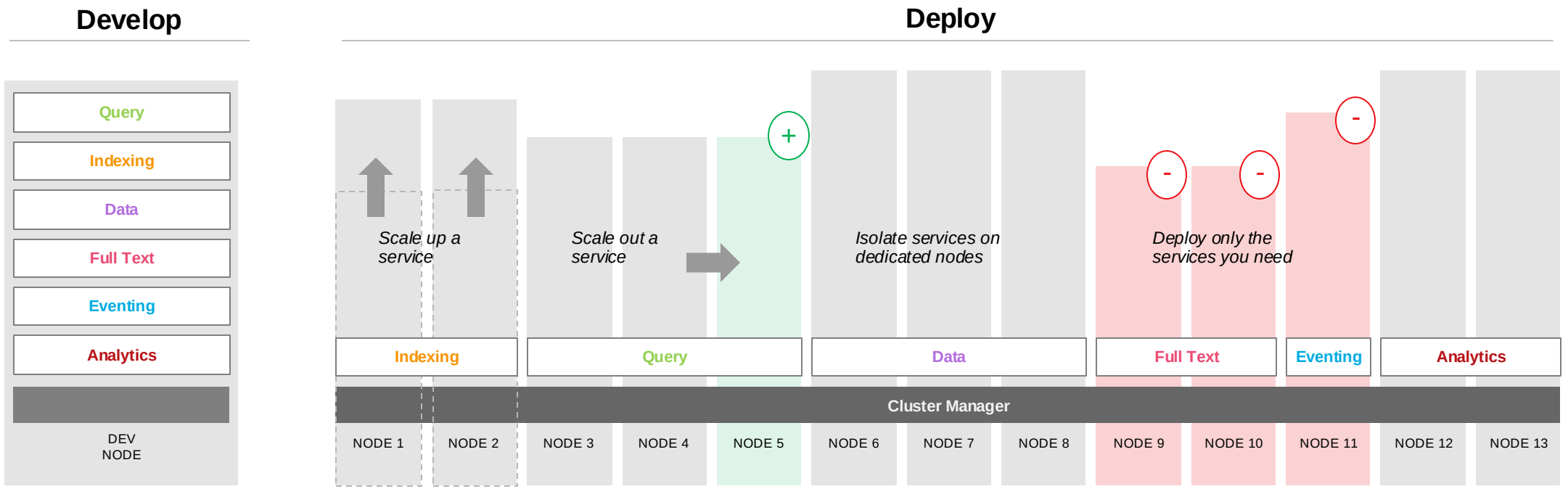
- **Key 기반 자동 분산 아키텍처**
  - 대량의 Key-Value 처리를 노드 별로 분산하여 성능 향상
  - 최대한 균등하게 분산 저장 가능, 특정 노드에 편중되는 현상(Data Skew) 방지
  - 별도의 분산 정책 불필요
- **Couchbase**
  - Key에 대한 Hash 알고리즘 적용으로 자동 분산
  - 노드 추가 시, Rebalancing을 통해 Key 재 분산 수행

- **Master Node-less 아키텍처**
  - 어플리케이션의 Key-Value Operation 시, 해당 Key에 매핑된 특정 노드에 직접 접근
  - 모든 노드가 어플리케이션 측면에서 Active 노드 역할
- **Couchbase**
  - 어플리케이션이 데이터 처리를 구현하기 위해 SDK 활용
  - 데이터 분산 정보(Cluster Map)를 지속적으로 SDK Client에 Update



2 아키텍처 : 자원 절약형 다차원 독립 확장

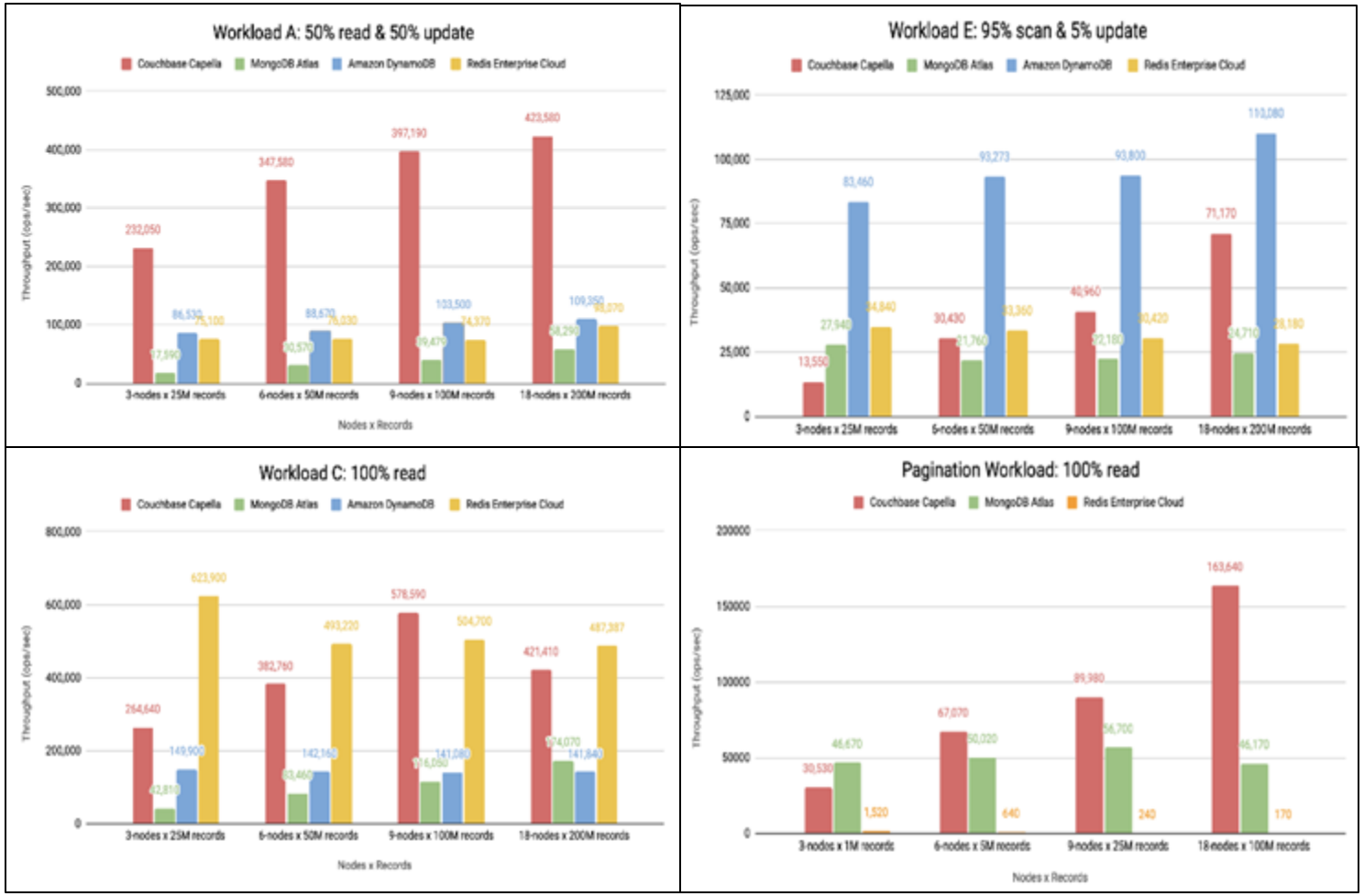
Couchbase는 Multi-Dimensional Scaling 기능으로 서비스 별 Workload 분산 및 독립성을 보장합니다.



- **Service 단위 하드웨어 자원 최적화**
  - 각 Service 별 시스템 자원을 독립적으로 할당
  - **각 Service에서 수행되는 작업이 다른 Service에 영향을 최소화**, 예를 들어 Analytics Service에서 복잡한 작업을 수행하여 시스템 자원을 많이 사용해도 Data Service 혹은 Query Service에서 수행하는 Operational 작업에는 영향이 없는 구조

3 성능 : NoSQL Database 벤치마크에서 탁월한 우위

<Yahoo! Cloud Serving Benchmark(YCSB-NoSQL Benchmark)>



출처 : https://www.altoros.com/blog/couchbase-capella-vs-mongodb-atlas-vs-amazon-dynamodb-vs-redis-enterprise-cloud/

Altoros, April 2023

MongoDB

- Struggles to scale, strongest at 3 nodes
- Price performance makes it worse
- Weakest performer

DynamoDB

- Excels at scans, but throws excessive errors
- Challenged across other workloads

Redis

- Excels at read-only, Capella meets at scale
- Regularly used as cache for Atlas or DynamoDB
- “Fails” Pagination workload

Capella

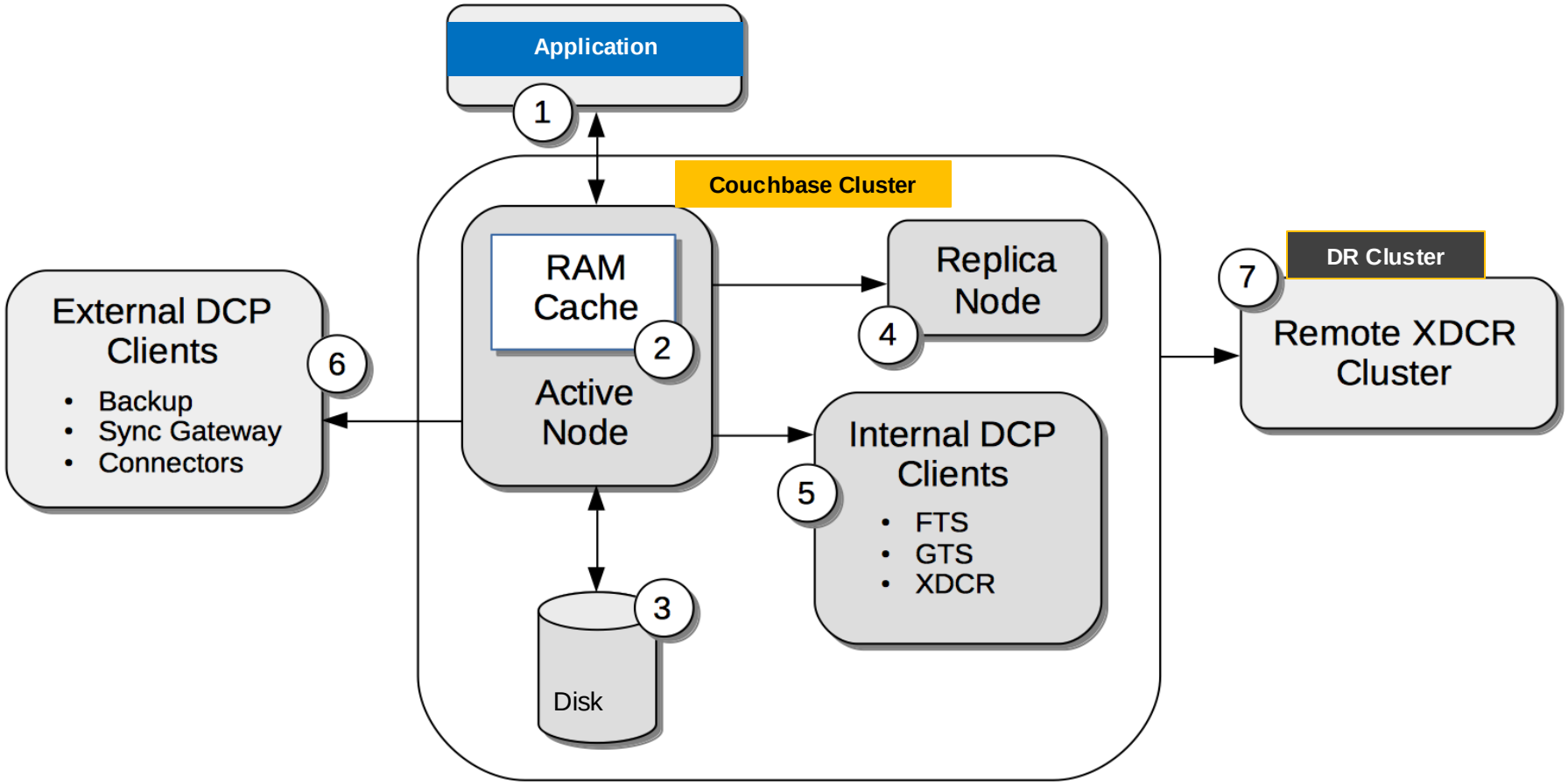
- Excels at Read and Write
- Scales effectively for multiple workloads
- Best price-performance

	Couchbase Capella	MongoDB Atlas	Azure CosmosDB\ (MongoDB API)
	90 /116	84 /116	71 /116
DEPLOYMENT	28 /32	27 /32	21 /32
MANAGEMENT	19 /25	19 /25	19 /25
SUPPORT	11 /16	12 /16	9 /16
PERFORMANCE	20 /26	13 /26	17 /26
PRICING	12 /17	13 /17	5 /17

출처 : https://benchant.com/navigator/dbaas

3 성능 : End-to-End 압축으로 네트워크/스토리지 비용 절감

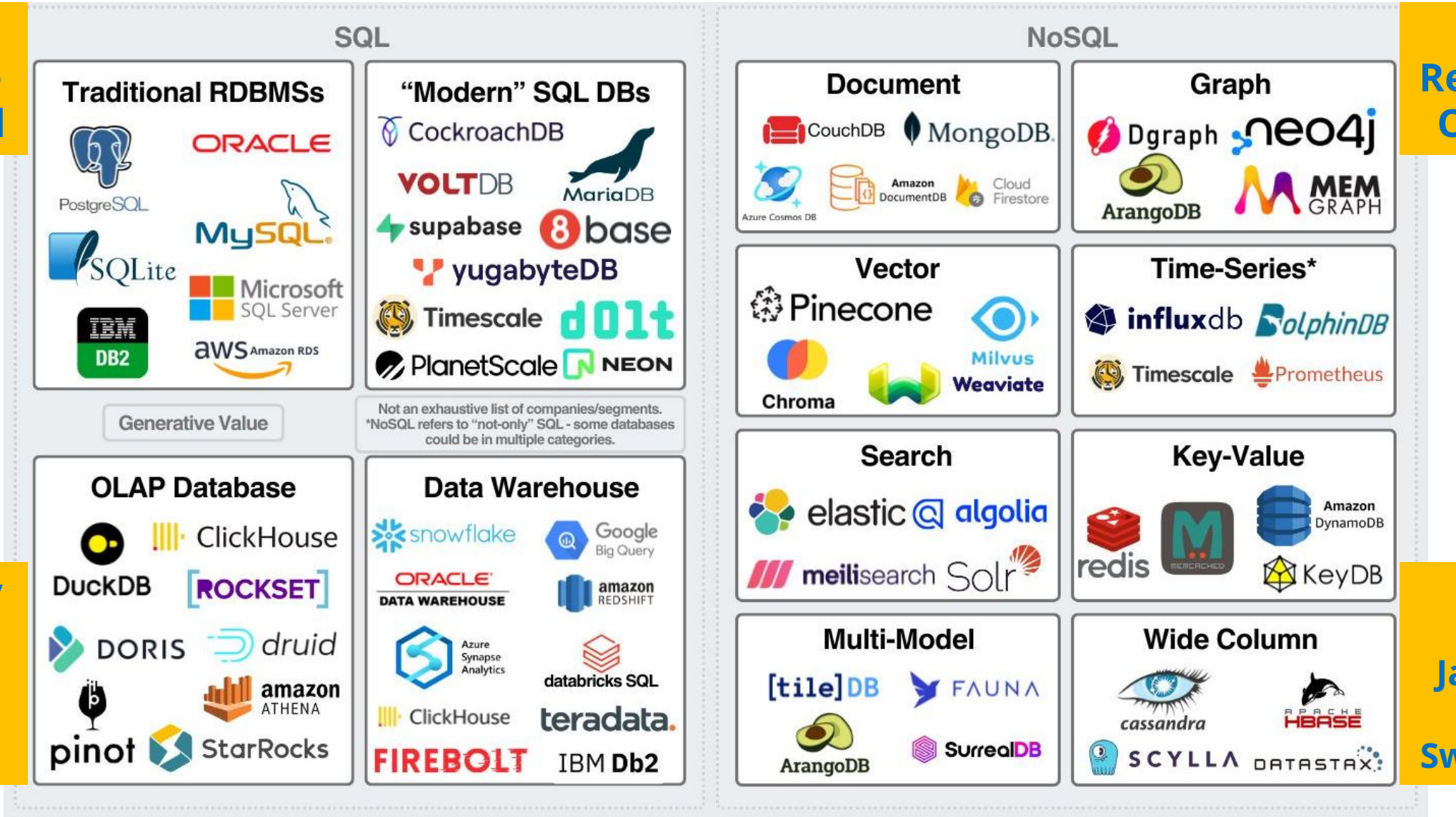
Couchbase는 데이터를 압축하여 전송하며 압축된 형태로 메모리와 디스크에 보관됩니다. 즉, 네트워크 전송량, 메모리 및 디스크 사용량을 줄려 성능 및 운영비용을 절감할 수 있습니다.



4 데이터 모델 : SQL(Table)과 NoSQL(Real World)

1970 Machine Oriented

Assembly Cobol C Pascal Perl



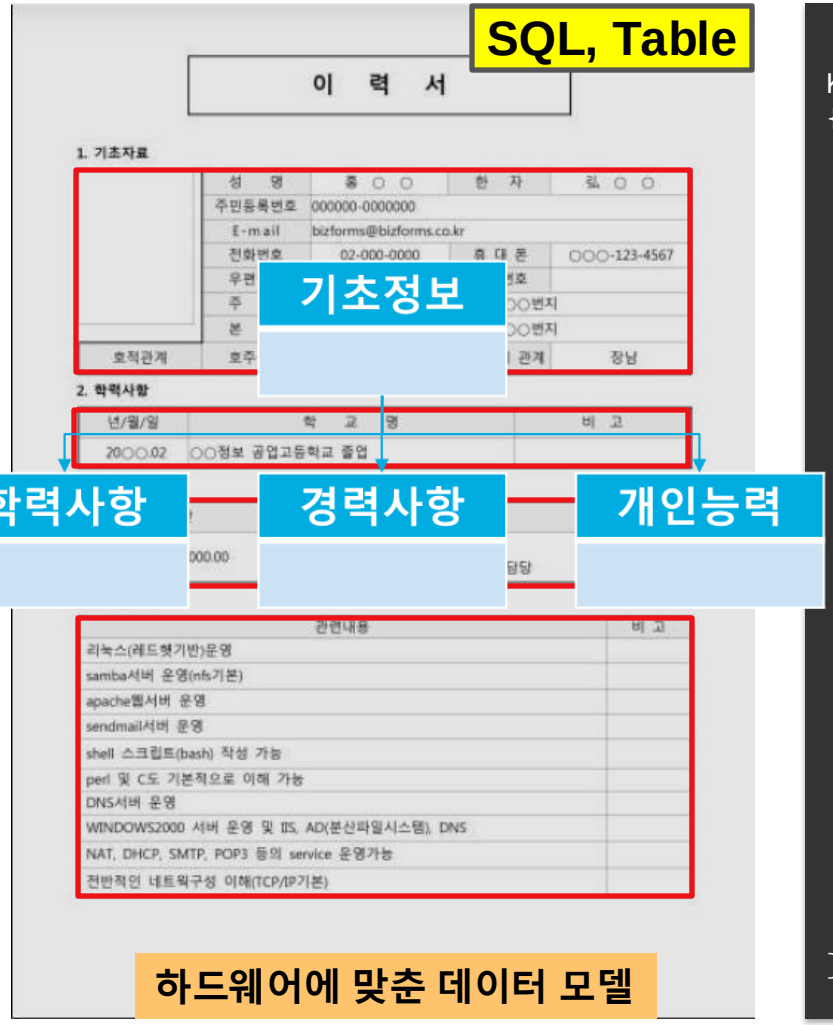
2020 Real World Oriented

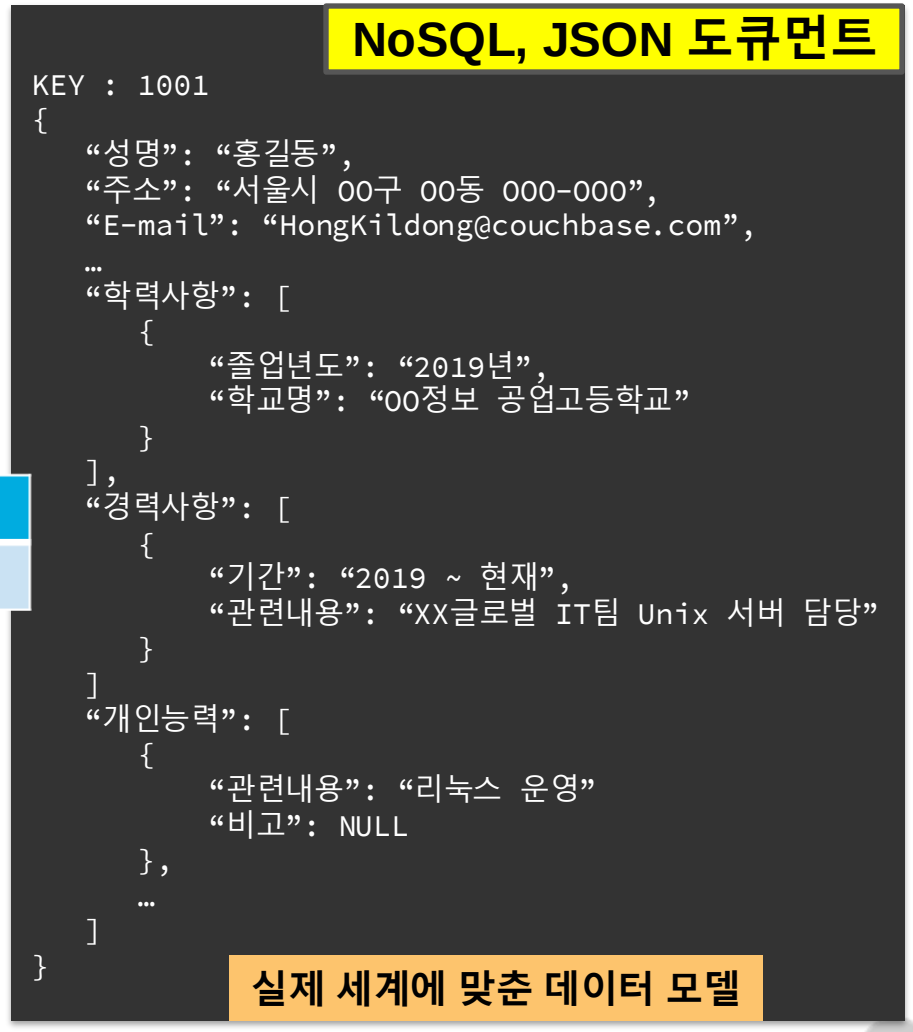
JAVA C# JavaScript Python Swift/Kotlin

4 데이터 모델 : JSON 도큐먼트

JSON은 텍스트로 이루어져 있으므로, 사람과 기계 모두 읽고 쓰기 쉽다. 프로그래밍 언어와 플랫폼에 독립적이므로, 서로 다른 시스템 간에 객체를 교환하기에 좋다.

- JSON 도큐먼트의 장점
  - 단일 도큐먼트 내에 다양한 정보를 계층 구조를 활용하여 저장
  - 정보 추가/삭제가 유연한 구조 제공
  - 데이터 전달을 위한 표준 인터페이스 역할
- RDB와 차별점
  - 여러 테이블로 분리, 저장되는 데이터를 단일 도큐먼트에 저장
  - 테이블 간 조인을 최소화하여 데이터 처리 속도 향상





4 데이터 모델 : ANSI 표준 SQL vs. 자체 Query 언어

2018년 Q4 고객에 대해서 각 미팅에 소요된 시간, 해당 고객에 모든 미팅에 소요된 시간 대비 해당 미팅의 소요 시간 비율, 순위 등을 조회하는 Query에 대해 Couchbase와 타 NoSQL 솔루션 비교입니다.

```
SELECT
  c.name,
  a.title,
  a.actduration,
  a.startDate,
  SUM(a.actduration) OVER ( PARTITION BY
    c.name ORDER BYc.name, a.startDate )running_total,
  TRUNC(100*(a.actduration/SUM(a.actduration)
      OVER(PARTITION BYc.name)) ) pct_of_total_time,
  RANK()OVER(PARTITION BY c.name ORDER BY
    (ARRAY_COUNT(a.contacts)/
     ARRAY_COUNT(c.contacts))DESC)hightouch_rank
FROM activity a
  INNER JOIN account c
    ON (a.accid = c.id)
WHERE a.activityType = 'Appointment'
   AND a.startDate BETWEEN '2018-10' AND '2018-12'
GROUP BY c.name,a.title,a.startDate,
         a.actduration,a.contacts,c.contacts
ORDER BY c.name,a.startDate
```

21 lines vs **347** lines

```
db.activity.aggregate([
  { $match : { type: "activity" } },
  { $match : { activityType:"Appointment" } },
  { $match : { startDate:{$gt:'2018-10-01',
                           $lt:'2018-12-31' }}},
  { $lookup: {
              from: "account",
              localField: "accid",
              foreignField: "id",
              as: "account_docs"} },
  { $match : { "account_docs": {$ne: [] } }},
  { $unwind: "$account_docs" },
  { $group : { "_id": {
      name: "$account_docs.name",
      title: "$title",
      startDate: "$startDate",
      duration: "$actduration",
      activity_contacts: "$contacts",
      account_contacts: "$account_docs.contacts",
    }}},
  { $addFields: { total_time: total_time } },
  { $addFields: { hightouch_rank: rank_temp } },
  { $addFields:{ running_total: running_total}},
  { $project: {
       "_id": 0,
      name: "$_id.name",
      title: "$_id.title",
      startDate: "$_id.startDate",
      duration: "$_id.duration",
      activity_contacts:
           "$_id.activity_contacts",
      account_contacts:.....
```

- **Couchbase**
  - ANSI 기반 SQL
  - Join, Windows Function 등 지원

4 데이터 모델 : ANSI 표준 SQL++

- 개요
  - **ANSI 2003 SQL 기반**
  - Multi-core 병렬 수행에 최적화된 Query Engine
  - **Cost-based Optimizer 지원**
- 확장 SQL
  - NEST/UNNEST : Embedded Object, Arrays 지원
  - IS EMPTY/IS MISSING : Flexible Schema
- SQL-Like
  - 99 % 표준 SQL과 동일한 Syntax
  - DML 지원 : Insert/Select/Update/Delete/Upsert
  - **INNER/OUTER JOIN 지원**

Power of SQL

Flexibility of JSON

```
{
    "Name" : "Jane Smith",
    "DOB"  : "1990-01-30",
    "Billing" : [
        {
            "type"    : "visa",
            "cardnum" : "5827-2842-2847-3222",
            "expiry"  : "2025-03"
        },
        {
            "type"    : "master",
            "cardnum" : "6274-2542-5847-3949",
            "expiry"  : "2024-12"
        }
    ]
}
```

4 데이터 모델 : 논리 / 물리 모델

- RDBMS와 유사한 구조의 논리 계층 구조로 구성하여 편리한 데이터 관리
- Data 서비스를 완전 메모리DB로 사용도 가능하며 용도에 따라 물리 저장 방식을 선택할 수 있음

RDBMS	Couchbase
Server	Cluster
Database	**Bucket**
Schema	**Scope**
Table	**Collection**
Row	**Document (JSON)**
Value	Sub-Document, Array

Feature	Ephemeral Bucket	Couchbase Bucket	Magma Bucket
Bucket memory quota (per node)	Min 256MB	Min 256MB	Min 1024MB
Max Object Size	20MB	20MB	20MB
Persistence	no	yes	yes
Replication and XDCR	yes	yes	yes
Encrypted data access	yes	yes	yes
Rebalance	yes	yes	yes
N1QL, Seach, Analytics, Eventing	yes	yes	yes
Indexing	yes	yes	yes
Backup	yes	yes	yes

5 서비스 : Capella Columnar Service : Cloud Data Lake

내부 JSON 데이터를 비롯하여 다양한 외부 데이터 소스를 통합하여 분석을 수행하는 컬럼 기반 Data Lake 기능 출시

Real-time Adaptive Applications (Operational, Transactions, Analytics, AI)

Couchbase CAPELLA

Operational Services

Key-Value, ANSI SQL, Full Text Search

Magma KV

JSON

Columnar Services

Independent Compute
MPP Query Engine

Independent Storage
Distributed, Columnar

Kafka

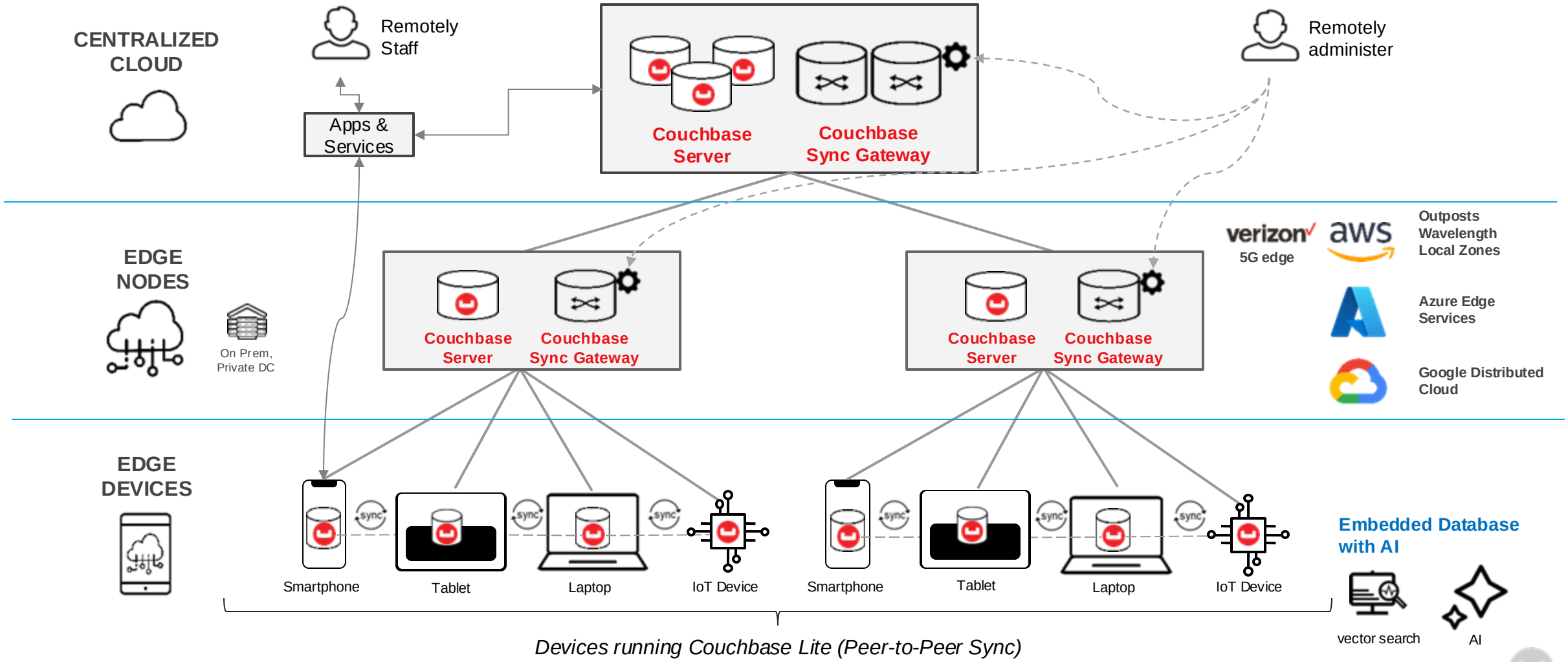
Object Store Connector

Other data stores
NoSQL, RDBMS,
Streaming

S3 Files
JSON,CSV, TSV,
Avro, Parquet,...

5 서비스 : Mobile (Couchbase Lite, Sync Gateway)

데이터 관리가 필요한 모든 디바이스에 데이터베이스를 적용할 수 있으며, 데이터 센터의 데이터베이스와 손쉬운 일관성을 유지



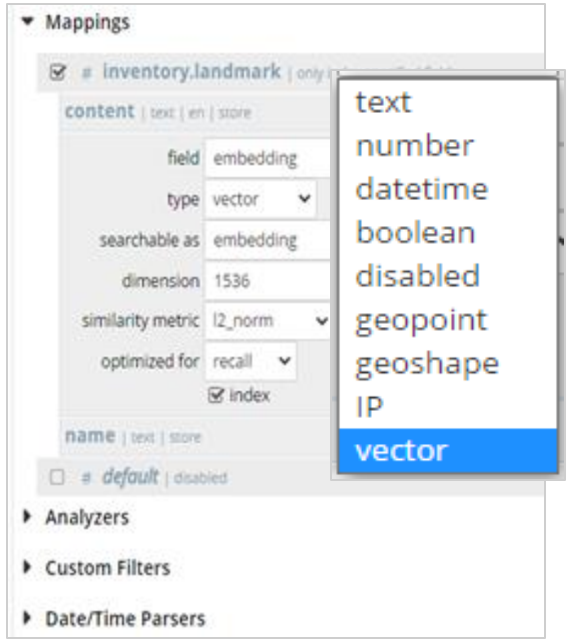
5 서비스 : Full Text Search with AI 벡터 검색

SQL 쿼리에 Full Text 검색 쿼리와 벡터 검색 쿼리를 통합 가능

JSON Storage

```
{ "type": “shoes",
 "productId": "CP123456",
 "category": “Gym Shoes",
 "name": “Beach Sneakers",
 "brand": “Ultimate Surf",
},
  "description": “The ultimate companion for beach adventurers, designed to seamlessly transition from sandy shores to urban landscapes. This innovative sneaker features a water-resistant, quick-drying mesh upper, allowing your feet to breathe while keeping them dry.",
 “descriptionVector": [0.131, 0.339, -0.611,0 .981,...]
```

Indexes

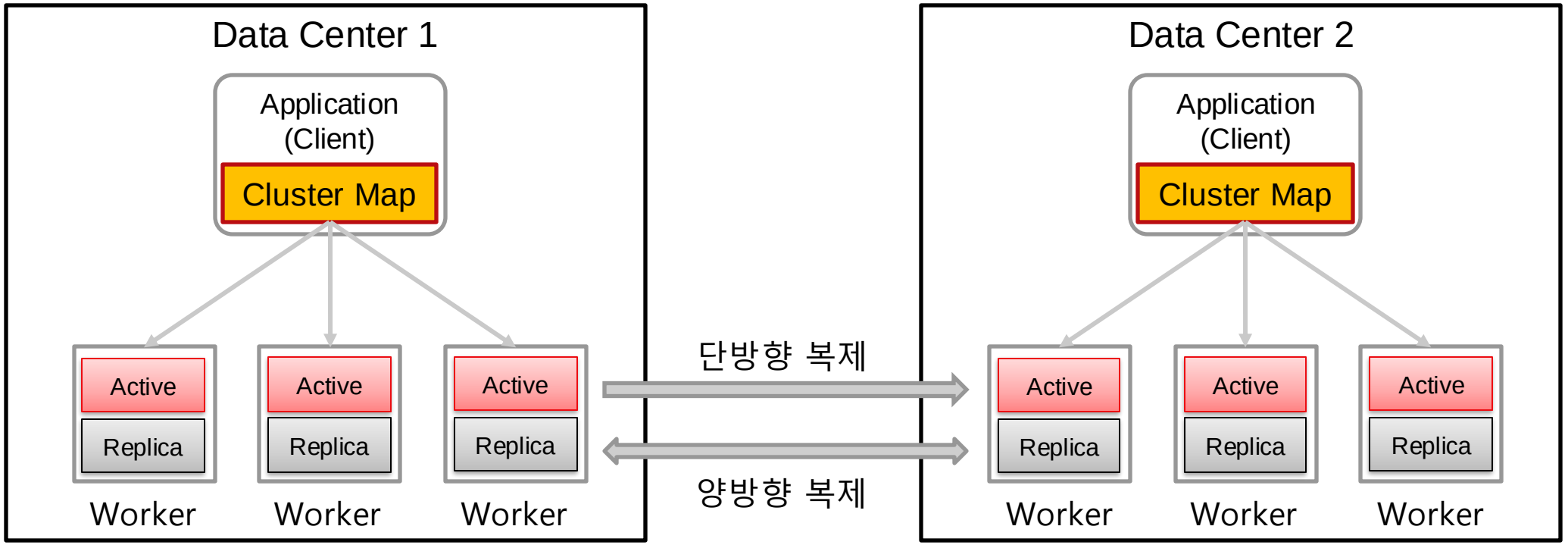


SQL + 자연어 검색 + AI Vector Search

```
SELECT meta().id, t.name
FROM `product` as t
WHERE
SEARCH(t,
 {“query”:{“match_phrase”: “sneaker”, “field”: “description”}}
)
ORDER BY search_score() DESC
LIMIT 10;
```

```
SELECT *
FROM product
WHERE LOWER(product.type) = ‘shoes’
AND product.size = 11
AND product.price between 50 and 80
/* desc SIMILAR TO  ‘blue running shoes’ */
ORDER BY GSI_VECTOR_ORDER(desc_embedding, {
  "knn": [{
    "field": "desc_embedding",
    "vector": [0.1, 0.334, -0.604, 0.985]  }]}
LIMIT 4
```

6 가용성 : 클러스터 간 Native 복제를 통한 재해 복구



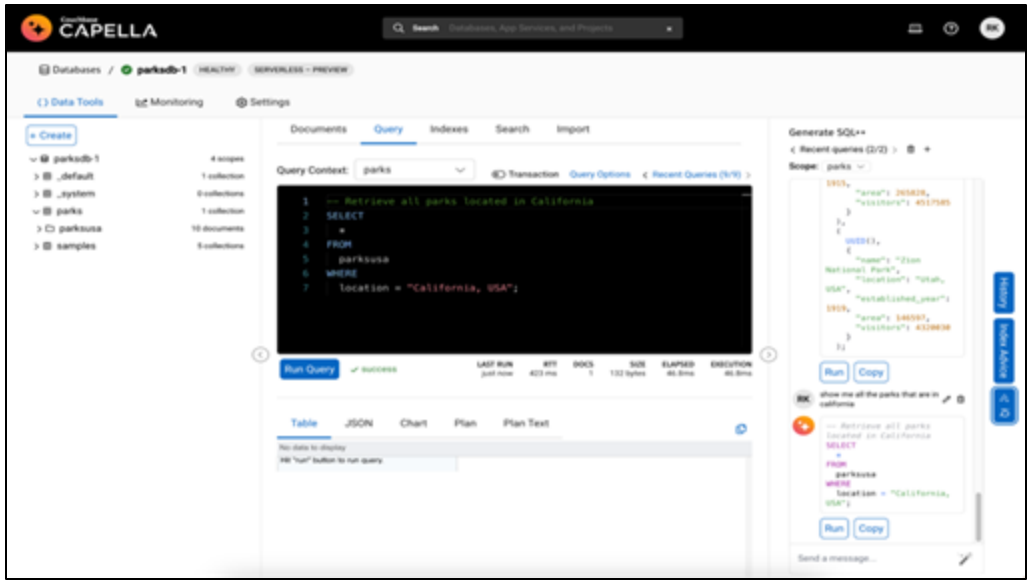
- **XDCR(Cross Data Center Replication)**
  - XDCR을 통해 멀티 데이터 센터에 위치한 클러스터 간 데이터 복제
  - 단방향 복제 및 양방향 복제 지원
  - 복제는 필요한 데이터만 필터링 가능
  - 단순 재해 복구 솔루션 이상의 글로벌 워크로드 분산 솔루션

7 개발 편의성 : 생성형 AI 기반 코딩 지원

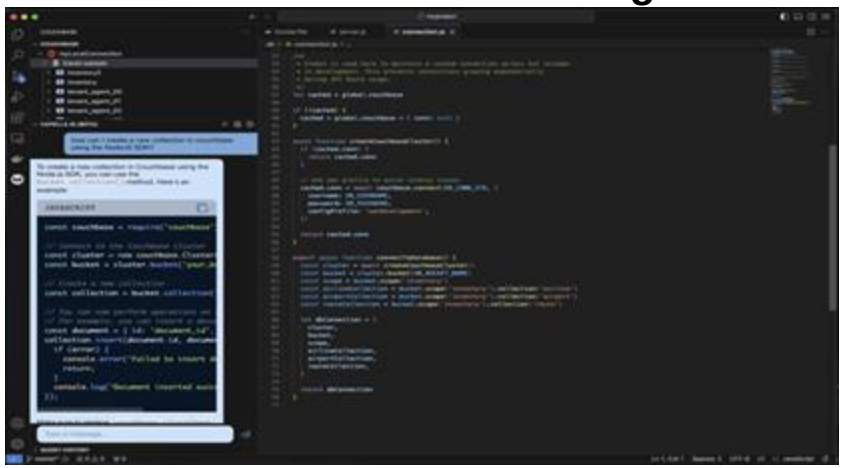
- Generative AI의 LLM을 활용한 Couchbase Capella 전용 Code Assistant
- 자연어로 SQL 및 소스 코딩 지원
- Couchbase 내부 스키마 정보를 활용하여 실제적인 코딩 지원

Couchbase CAPELLA | iQ

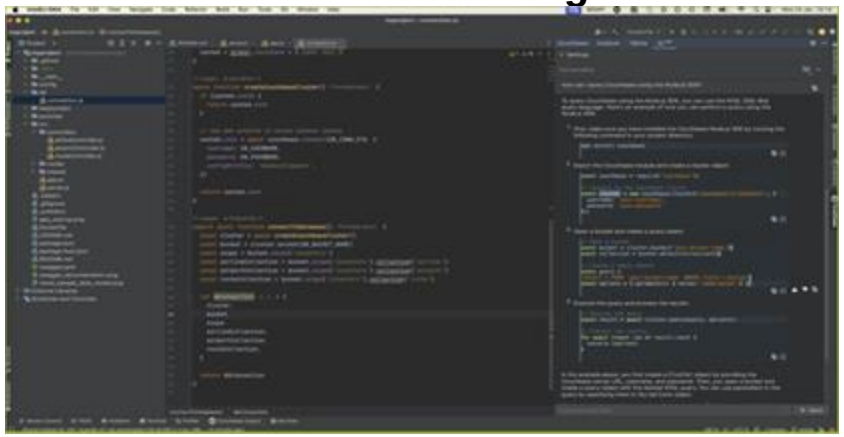
Capella Workbench



Visual Studio Code Plug-in



IntelliJ IDEA Plug-in



8 설치/구성 : 지원 플랫폼

Bare-Metal, VM, Container 와 완전관리형 클라우드 데이터베이스 서비스(DBaaS) 사용 가능

Couchbase CAPELLA

Couchbase

Fully Managed

- **완전관리형 데이터베이스 서비스**
- **AWS, GCP, Azure**

- **설치, 구성, 모니터링, 업그레드 등 모든 운영은 Couchbase가 담당**

Enterprise

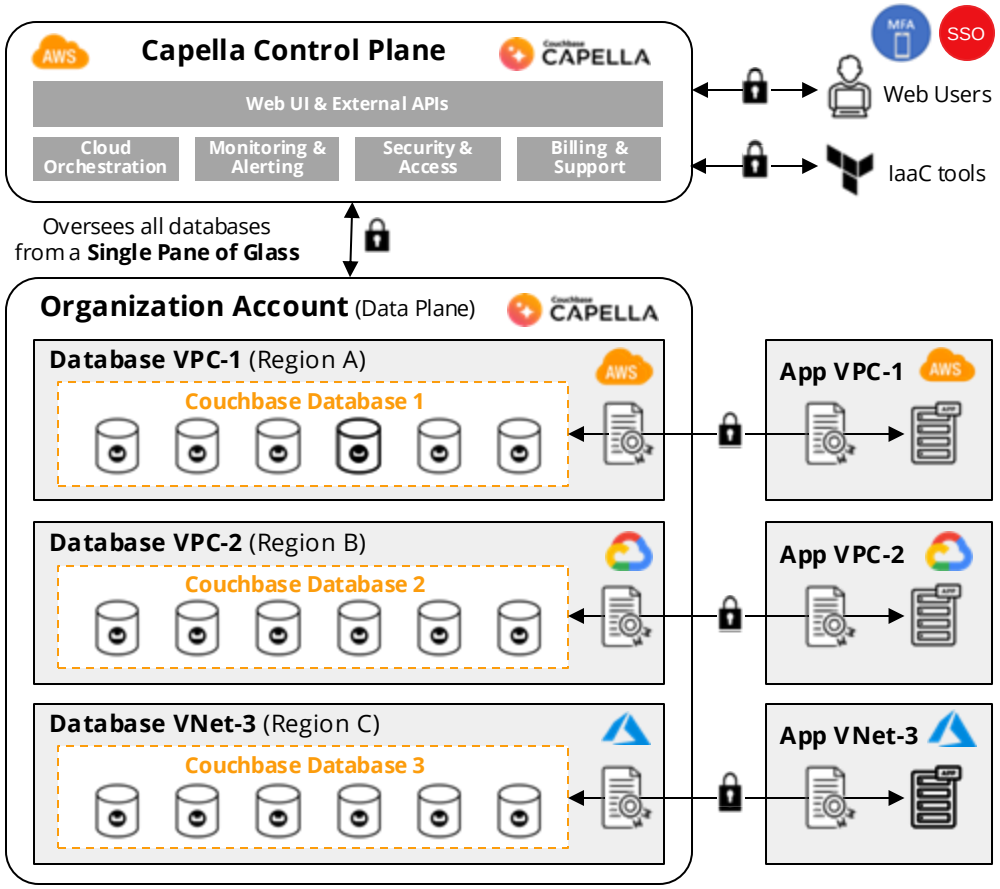
- Bare-metal 서버, 클라우드 IaaS 서버
- Private Cloud 서버, K8S 컨테이너

- 설치, 구성, 모니터링, 업그레드 등 모든 운영은 고객이 수행

Linux, Windows, MacOS, Intel/AMD, ARM

9 설치/구성 : 카펠라 아키텍처

완전관리형 클라우드 데이터베이스 서비스(DBaaS)인 Capella 아키텍처



Capella Control Plane

- **Manages** the Cloud Orchestration, Monitoring & Alerting, Security & Access, Billing & Support
- Is the Access Point for Organization **Web UI** Users
- Allows Infrastructure as Code Tools (e.g. **Terraform**)

Organization Account (Data Plane)

- **Account Isolation**: 1 Account per Organization
- **Database Isolation**: 1 VPC per Couchbase Database
- **Multiple Clouds**: AWS, Google and Azure

Applications

- **Connect** directly to Databases
- **Multiple** connectivity options: over Public Connection, through VPC Peering or Private Link
- All communications are **encrypted**

10 요약하면,

Couchbase는

JSON Document 직관적이고

- 사람이 인지 하는 세상을 그대로 데이터 모델로 사용
- 복잡한 정규화 과정없이 직관적인 방식으로 어플리케이션 개발/운영

SQL, Generative AI 익숙하고, 쉽게

- NoSQL 이지만 표준 SQL을 지원
- 생성적 AI인 Capella IQ 지원으로 더 손쉬운 개발이 가능

Data Platform 일관적 적용

- Key/Value 데이터서비스에서 분석서비스, 모바일 앱서비스까지 일관성있게 업무에 적용 가능
- 센서, 모바일, 퍼스널컴퓨터, 데이터센터 서버, 쿠버네티스, 클라우드에 동일한 데이터플랫폼 적용으로 개발의 일관성 뿐만 아니라 데이터의 일관성도 보장

Enterprise 에서 요구하는 성능, 안정성, 보안성

Vector Search 시리즈 1,2 내용

1. Introduce AI
2. Demystifying Vector Search
3. The power of Hybrid Search
4. Vector Search Use Case : Semantic Search
5. Generative AI(LLM) & RAG

1. Adaptive apps at the edge
2. Vector Search at the edge
3. Dynamically Generating Embeddings
4. Visual Search

Adaptive apps at the edge

Edge AI Benefits

인터넷 사각지대에서 작동할 때 AI 기반 앱의 속도와 가동 시간을 어떻게 보장합니까?

Edge AI Architecture

- 데이터와 AI 모델 처리를 네트워크의 가까운 쪽(상호 작용 지점에 더 가까운 쪽)으로 가져옴
- 가능한 경우 에지 클라이언트에 데이터베이스와 가벼운 AI 모델을 내장하여 먼 클라우드 데이터 센터에 대한 종속성을 줄임
- 일관성과 무결성을 위해 에지와 클라우드 간에 데이터 동기화

The Promise Of Edge AI

Speed | **Reliability** | **Privacy** | **Bandwidth**

Couchbase Mobile | Product Stack

Cloud to Edge to Device

Couchbase Server

Scalable, performant NoSQL data platform

Sync Gateway

Secure gateway for data sync and access control

Couchbase Lite

Full featured NoSQL embedded database

Deploy Fully Managed

Couchbase CAPELLA

aws Azure Google Cloud

Deploy Self-Managed

Public Cloud

aws Azure Google Cloud

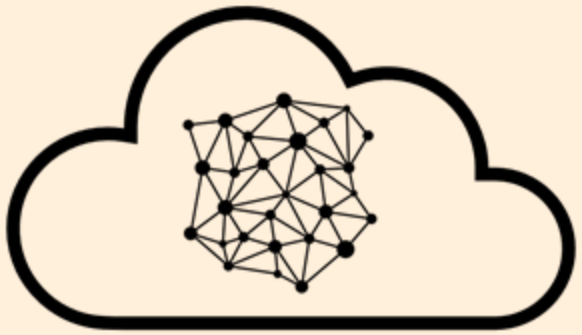
Private Cloud

Cloud Edge

verizon aws Outposts

On Prem

Couchbase Capella Is AI-Ready



COLUMNAR SERVICE

Aggregation of data across large data sets

Multiple sources: Capella, RDBMS, NoSQL, S3, Parquet

Fast processing for speedy ML training iterations

JSON accommodates multiple types and formats of data

Python UDF for calling trained ML models

VECTOR SEARCH

Similarity

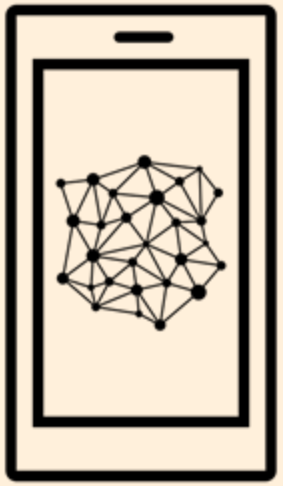
Semantic search

Retrieval Augmented Generation

Hybrid-search (vector, FTS, range, predicate, aggregations)

Couchbase Lite Is AI-Ready

Embedded database for mobile and IoT devices



PREDICTIVE QUERY API

Call optimized ML models to make predictions on Couchbase Lite data.

OFFLINE-FIRST

VECTOR SEARCH

Similarity

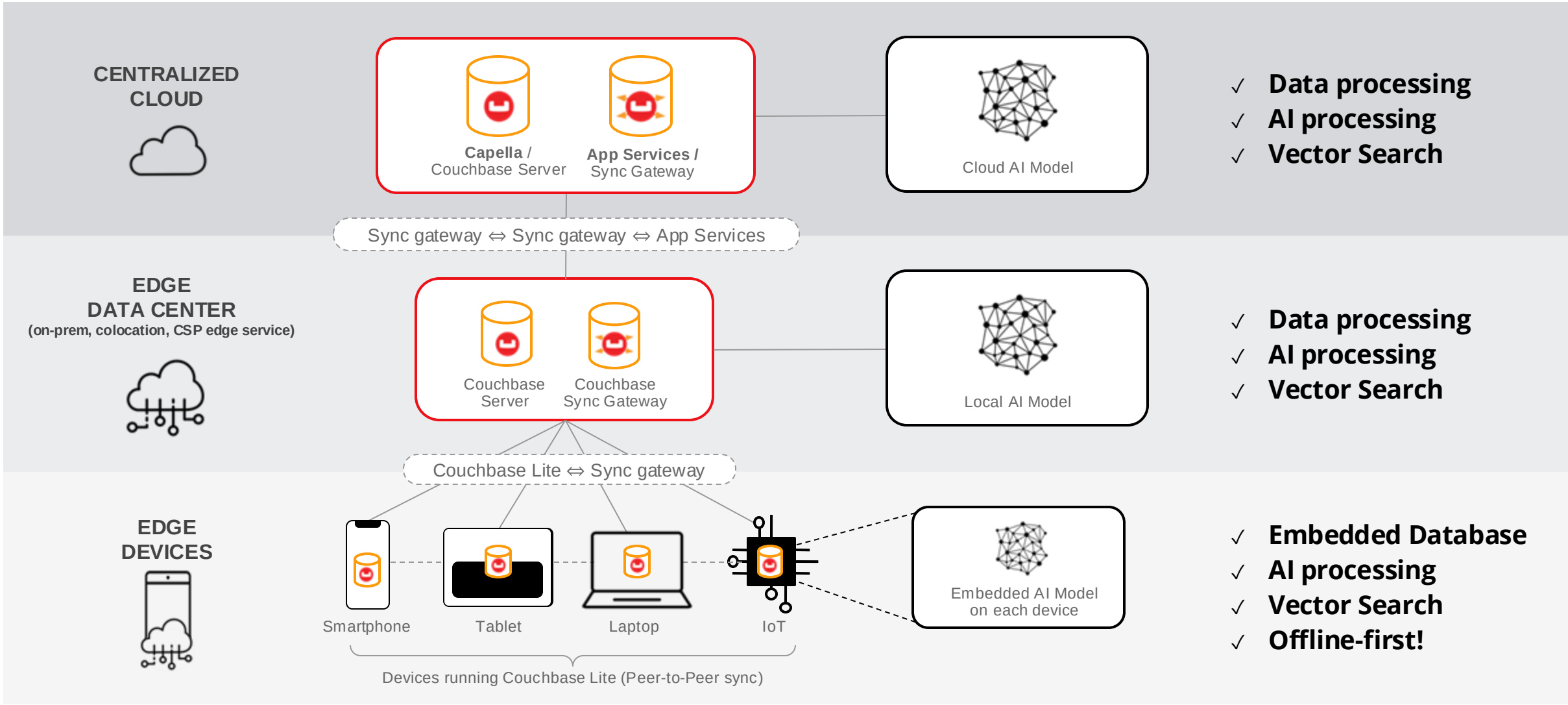
Semantic search

Retrieval Augmented Generation

Hybrid-search (vector, FTS, range, predicate, aggregations)

OFFLINE-FIRST

Couchbase Mobile | Cloud to Edge AI



Vector Search at the Edge

Benefits & Use Cases

Couchbase

Vector Search at the Edge | Benefits

Offline-first Mode

인터넷 연결 여부에 관계없이 항상 애플리케이션에서 의미 검색이 가능합니다.

Reduced Cost Per Query

데이터 전송 비용, 클라우드 AI 모델 비용 및 운영 비용을 절감하세요

Data Privacy at the Edge

개인 정보를 침해하지 않고도 RAG를 엣지에 적용하여 검색 결과를 사용자 지정하세요

Low Latency & Timely Response

로컬 인덱스와 내장 모델을 사용하여 로컬 데이터 세트에 대한 빠른 조회 검색

Unified Cloud to Edge Support

웹 백엔드 애플리케이션과 엣지 애플리케이션에서 유사성 검색 수행

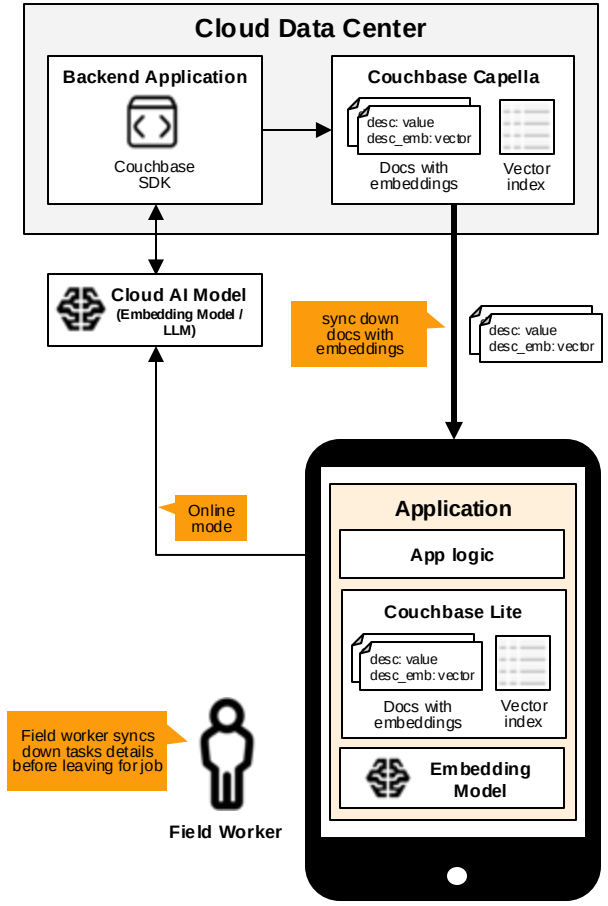
Offline-First Mode

Benefits

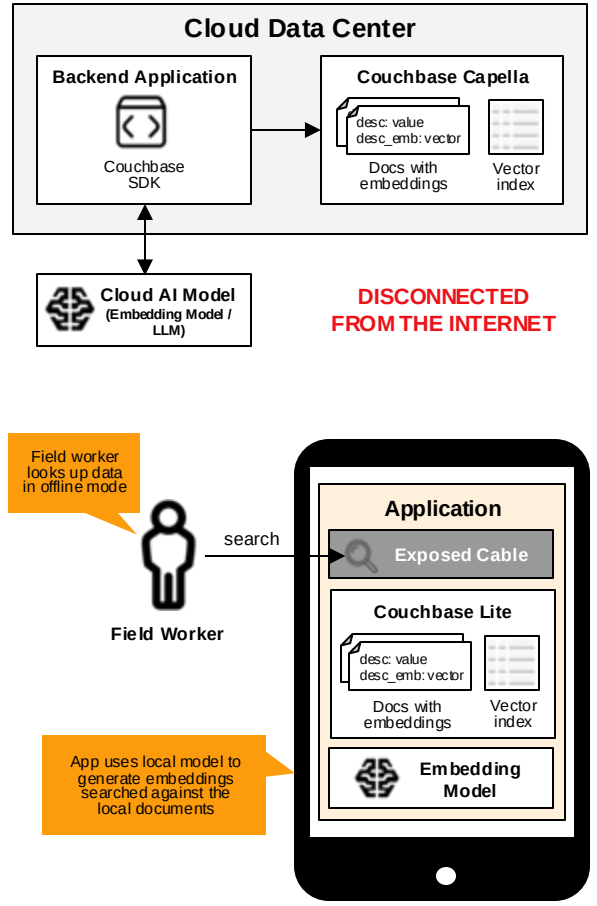
- **인터넷 연결 여부와 관계없이** 애플리케이션에서 의미 검색이 항상 가능합니다.
- 텍스트 기반 검색은 **문맥적으로 관련성 있는** 데이터를 빠르게 검색하기에 충분하지 않습니다.

Example

- 수리 현장과 재난 지역에 있는 공익사업 종사자는 인터넷 연결이 좋지 않거나 전혀 없는 지역에서 작업합니다.
- "line", "cable", "wire"라는 단어는 공익사업 회사의 동의어입니다. 현장에서 공익사업 종사자가 "line"을 검색하면 "cable", "wire"가 포함된 문서도 나타나야 합니다.
- FTS를 사용하면 애플리케이션은 만들고, 관리하고, 유지하기 어려운 동의어 목록을 유지해야 합니다.
- 관련성도 중요합니다. 따라서 "전선 낙하 시 안전 절차"에 대한 질의는 전선 낙하, 전기 케이블, 고압선 등과 관련된 매뉴얼에 초점을 맞춰야 합니다.



ONLINE - Before Field Worker sets out for job



OFFLINE - When Field Worker are on the job

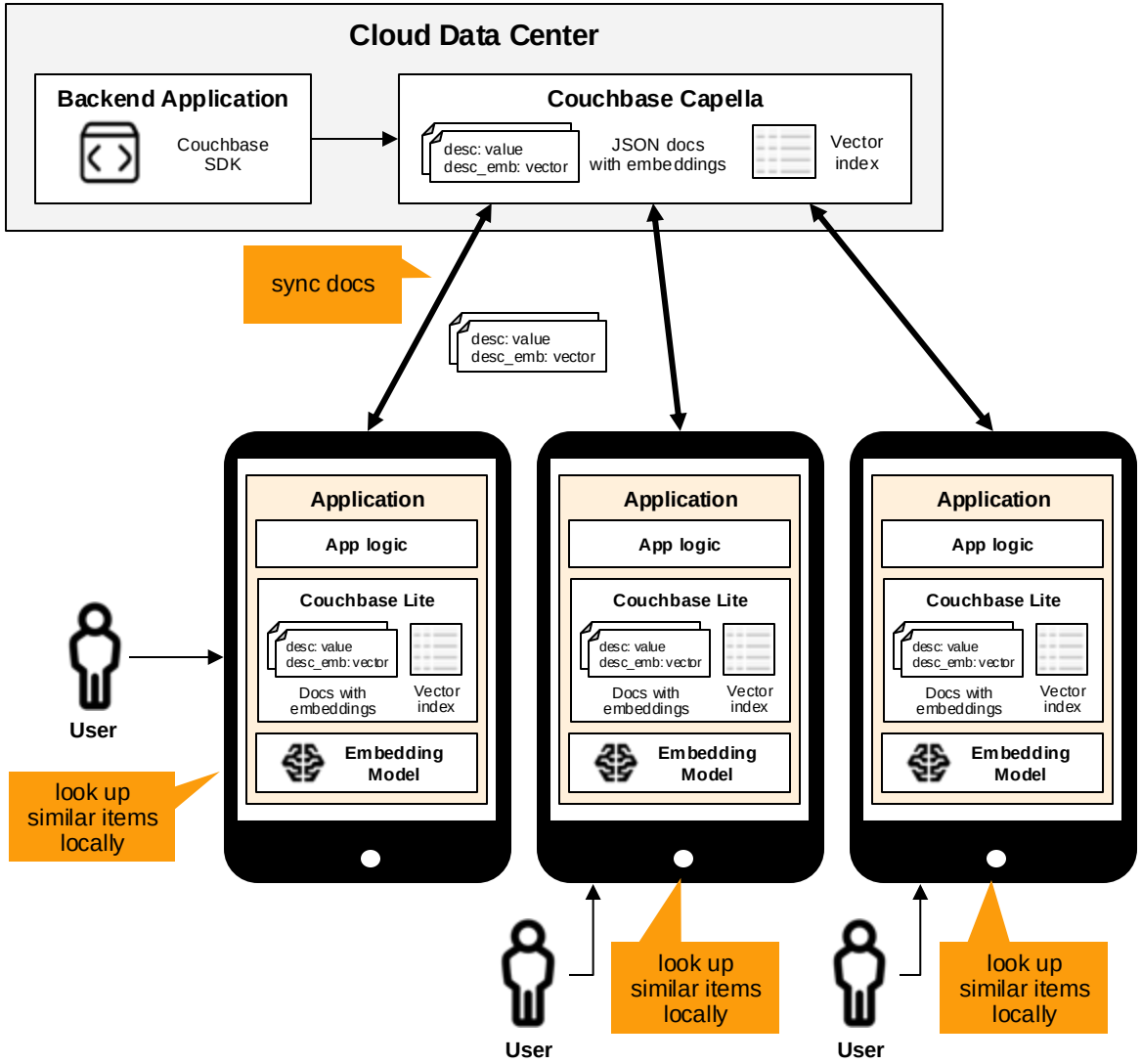
Reduced Cost Per Query

Benefits

- **데이터 전송 비용(클라우드 이탈 요금)을 줄이고** 엣지에서 로컬 모델을 사용하여 **대역폭을 절약**합니다. 엣지에서 생성된 데이터는 로컬에서 문서를 색인(인덱스) 함
- 연결된 수천 개의 모바일 기기에서 모든 쿼리가 클라우드에 도달하면 클라우드 모델의 부하가 높아질 것입니다.
- 또한 CPU에 대한 전기와 같은 **운영 비용을 기기에서 분산**합니다. 즉, 운영 비용은 최종 사용자가 밤에 휴대전화를 충전할 때 지불합니다.

Example

- 구독자가 수만 명인 매우 인기 있는 여행 모바일 앱은 사용자가 앱 내에서 예약 경험에 대한 리뷰를 제출할 수 있는 옵션을 제공합니다.
- 리뷰에 대한 감정 분석은 수십만 명의 고객이 원격 서버에 접속하는 대신 기기에서 로컬로 수행됩니다. 분석에 따라 사용자는 즉시 다음 단계로 안내됩니다. 예를 들어, 나쁜 리뷰는 사용자에게 고객 서비스 담당자와 통화할지 묻고, 긍정적인 리뷰는 사용자에게 감사를 표하며 경험을 맞춤화합니다.



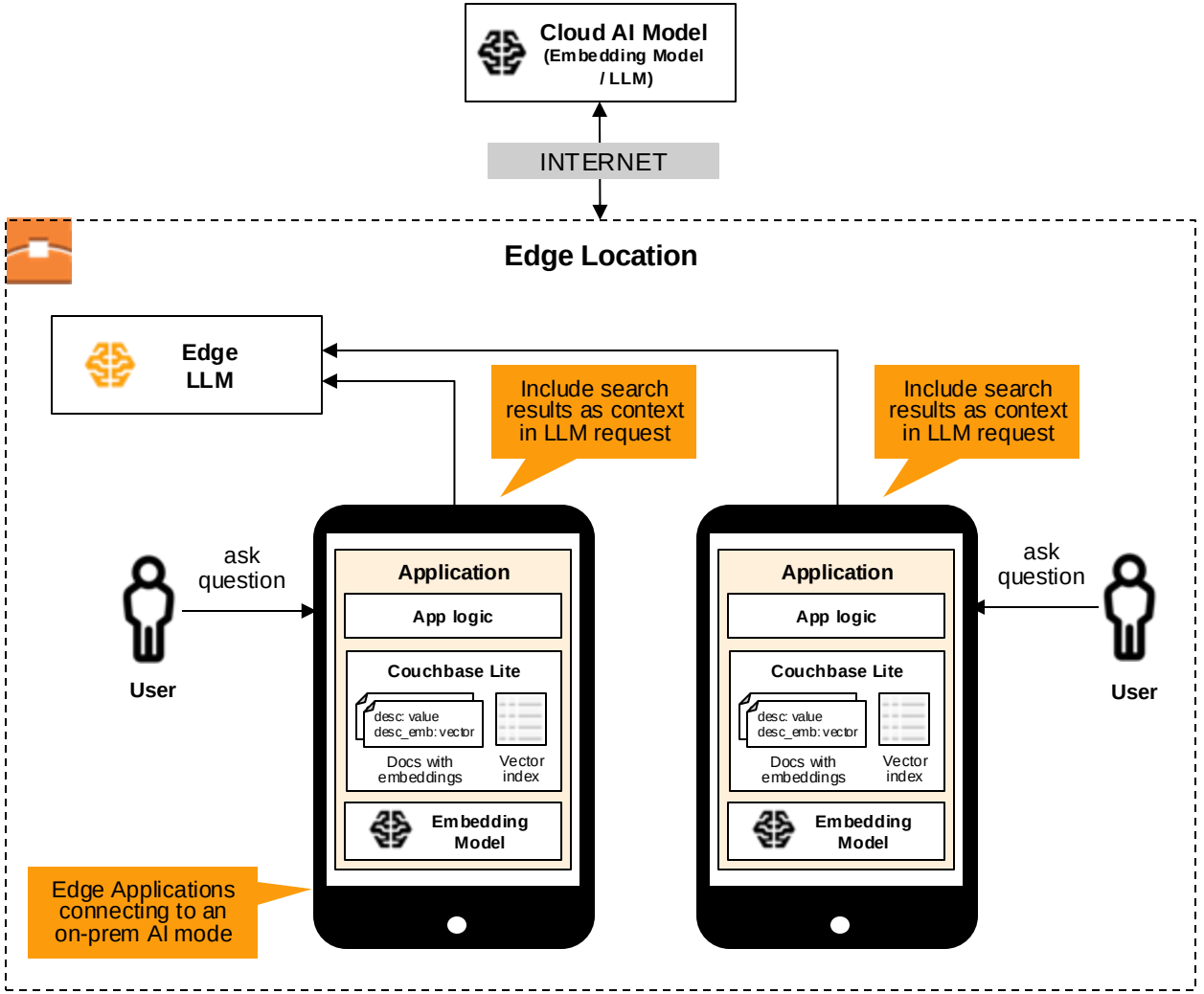
Data Privacy

Benefits

- 개인 데이터와 민감한 성격의 검색 쿼리는 장치나 엣지 위치를 벗어날 필요가 없습니다.
- **개인 정보를 침해하지 않고** 엣지에서 RAG로 검색 결과를 최적화하세요

Example

- 병원의 한 의사가 수술에서 회복 중인 환자를 위한 치료 옵션을 찾고 있습니다.
- 관련 환자 컨텍스트는 병력과 선호도에서 검색됩니다. 이 데이터에 대한 액세스는 인증되고 승인되므로 환자 또는 지정된 간병인만 액세스할 수 있습니다.
- 환자 컨텍스트는 사용자 지정 회복 계획을 생성할 수 있는 모델에 대한 RAG 입력으로 Edge LLM으로 전송됩니다.



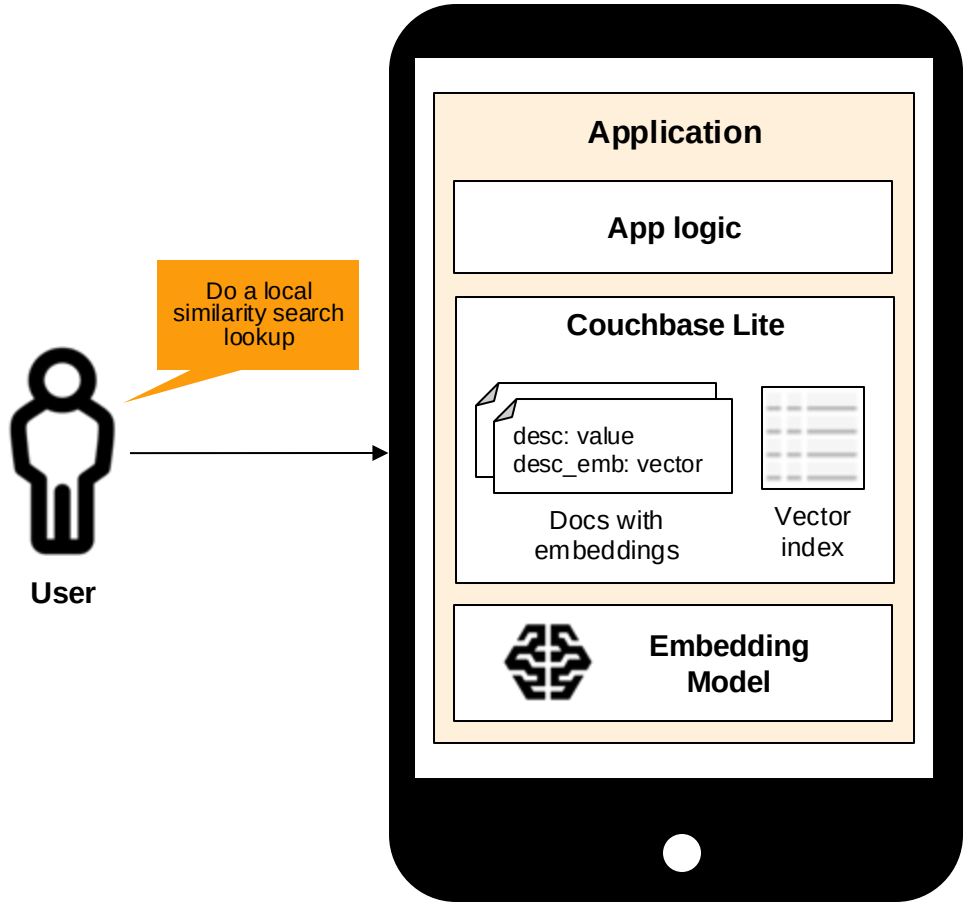
Low Latency & Timely Response

Benefits

- 로컬 인덱스와 임베디드 모델을 사용하여 로컬 데이터 세트에 대한 **빠른 조회 검색**
- 에지의 로컬 모델조차도 인터넷을 통한 클라우드 모델 액세스에 비해 RTT를 줄일 수 있습니다.

Example

- 소매점은 제품 카탈로그, 매장별 가격 및 프로모션 문서를 고객 서비스 지원 키오스크와 같은 에지 장치에 임베딩하여 동기화합니다. 문서는 키오스크에서 로컬로 색인됩니다.
- 키오스크의 사용자는 카메라로 촬영한 자신이 입고 있는 재킷과 어울리는 모자를 검색합니다. 사용자는 또한 세일 중인 모자에 관심이 있습니다.
- 각 소매점의 키오스크가 원격 서버로 검색 쿼리를 보내는 대신, 키오스크에서 카탈로그에서 로컬로 유사 검색을 수행하여 세일 중인 유사한 품목을 찾습니다.
- 또한 이미지를 키오스크에서 즉시 삭제할 수 있어 개인 정보 보호 문제가 완화됩니다.



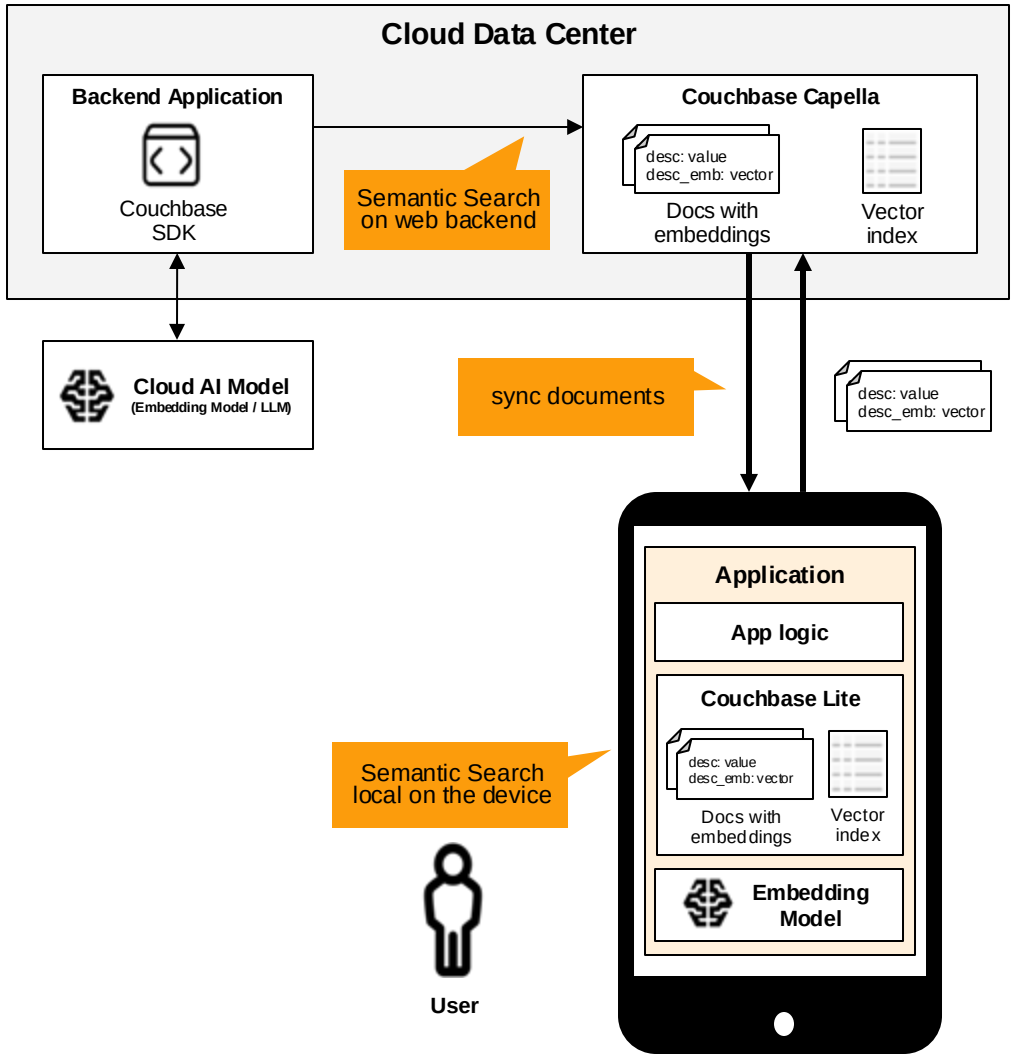
Unified Cloud to Edge Support

Benefits

- Capella/Couchbase 서버와 에지 장치 간에 동기화된 문서에 대해 클라우드와 에지에서 유사성 검색을 수행합니다.

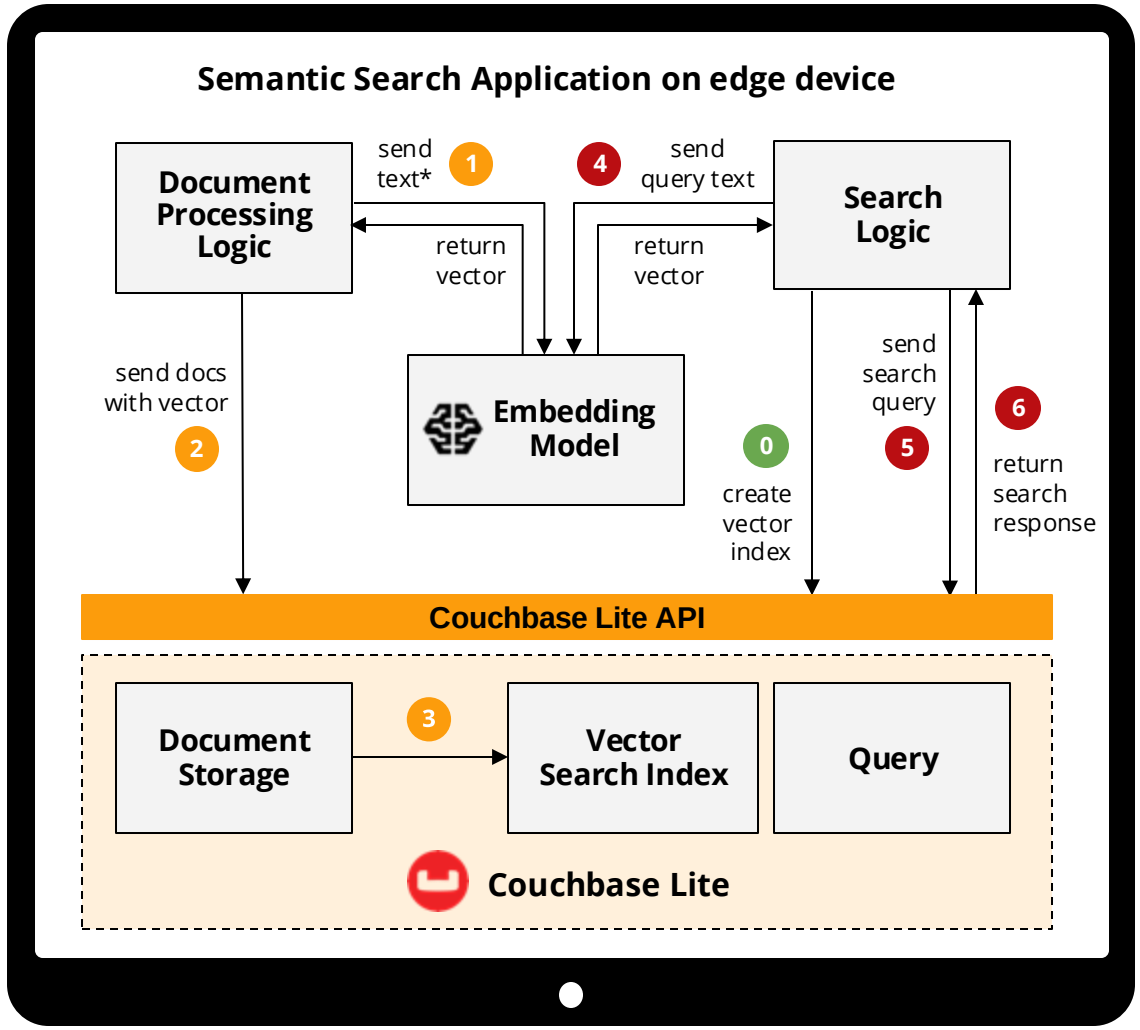
Example

- 지난 6개월 동안 사용자별 거래 내역이 동기화되어 기기에 로컬로 저장되는 모바일 뱅킹 앱을 생각해 보세요.
- 사용자가 몇 달 전에 구매한 것과 관련된 거래를 찾고 있습니다. 검색은 로컬에서 수행되므로 빠르고 오프라인에서도 사용할 수 있습니다.
- 모든 사용자와 관련된 거래는 클라우드 서버에 저장되며, 사기 탐지 애플리케이션에서 의미 검색을 사용하여 사기 활동의 패턴을 탐지합니다.



Semantic Search with Couchbase Lite

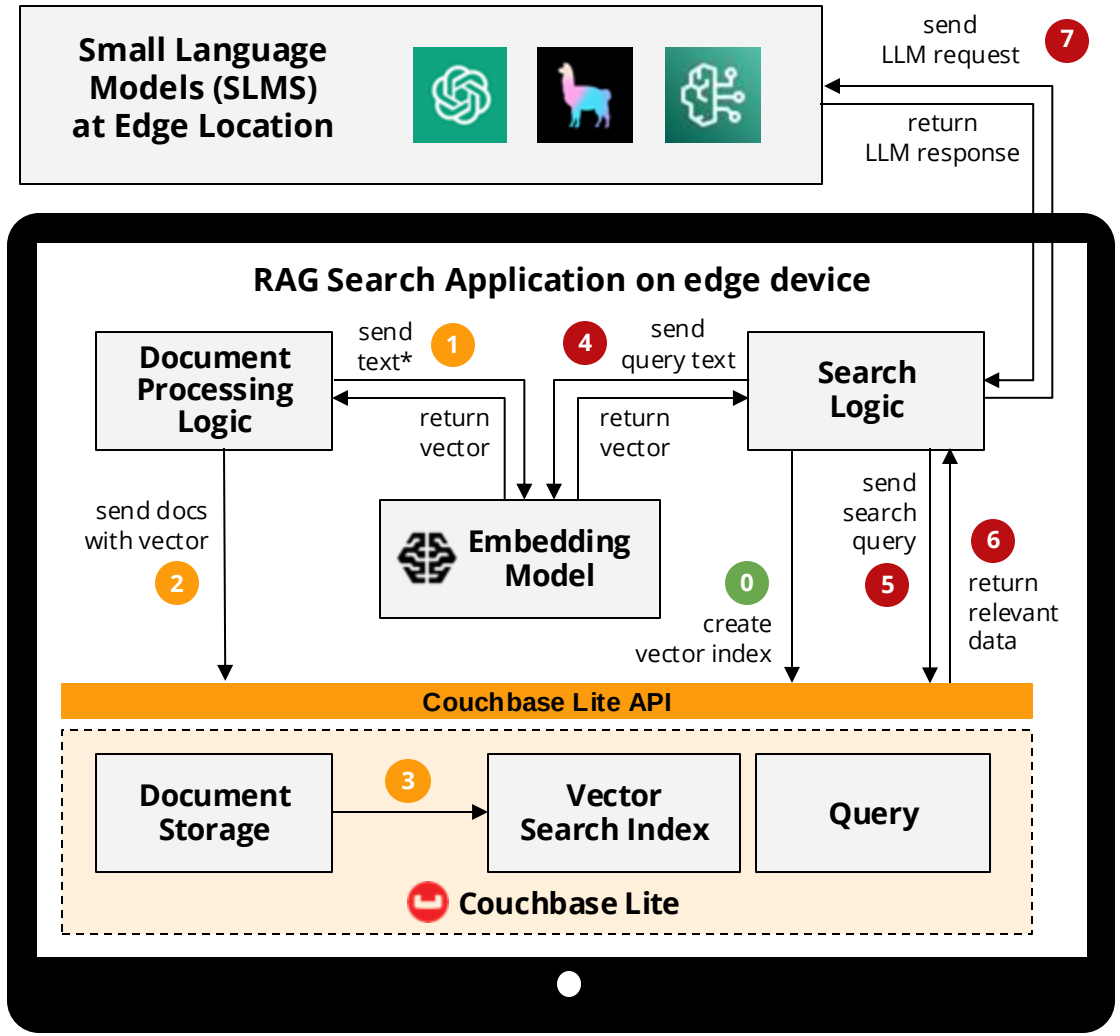
0. Application creates vector index
1. Application sends documents to embedding model that returns embedding vector. Application adds vectors to JSON documents
2. Application creates/updates JSON documents (text with vector) to Couchbase Lite
3. This triggers update to vector search index on Couchbase Lite that updates index with document
4. Application sends query text to embedding model. Embedding model returns corresponding embedding vector.
5. Application calls Couchbase Lite with a pure vector search, or hybrid mixed vector search and FTS, or hybrid vector and SQL++ query
6. Couchbase Lite returns top k-NN results of query with requested fields or objects - the vector search returns semantically similar documents



(*) can be any media

RAG with Couchbase Lite as Vector Database

0. Application creates vector index
1. Application sends documents to embedding model that returns embedding vector. Application adds vectors to JSON documents
2. Application creates/updates JSON documents (text with vector) to Couchbase Lite
3. This triggers update to vector search index on Couchbase Lite that updates index with document
4. Application sends “query text” to embedding model. Embedding model returns embedding vector
5. Application calls Couchbase Lite with a pure vector search, or hybrid mixed vector search and FTS, or hybrid vector and SQL++ query
6. Couchbase Lite returns top k-NN results of query with requested fields or objects
7. Application sends results of query as context to LLM for more accurate responses



(*) can be any media

Dynamically Generating Embeddings

Vector Search on documents without inline embeddings

Dynamically Generating Embeddings: Why?

앱에 의한 문서 변형 중에 문서 내에 임베딩을 포함하는 것이 항상 실용적이지 않을 수 있습니다.
임베딩은 AI 모델과 차원에 따라 상당히 커질 수 있습니다.

Storage Costs

모바일/에지 장치는 서버 장치에 비해 리소스가 제한적입니다.

Data Transfer Costs

클라우드와 임베딩을 사용하여 문서를 동기화하면 추가 대역폭이 발생합니다.

Local Embedded Model unavailable

오프라인 모드 장치에서 실행되는 앱은 로컬 임베딩 모델에 액세스할 수 없습니다.

Option 1: Using Prediction Function

Register Callback

- Application can register a callback `predict()` function with Couchbase Lite to dynamically generate vector embeddings

Predict() Function

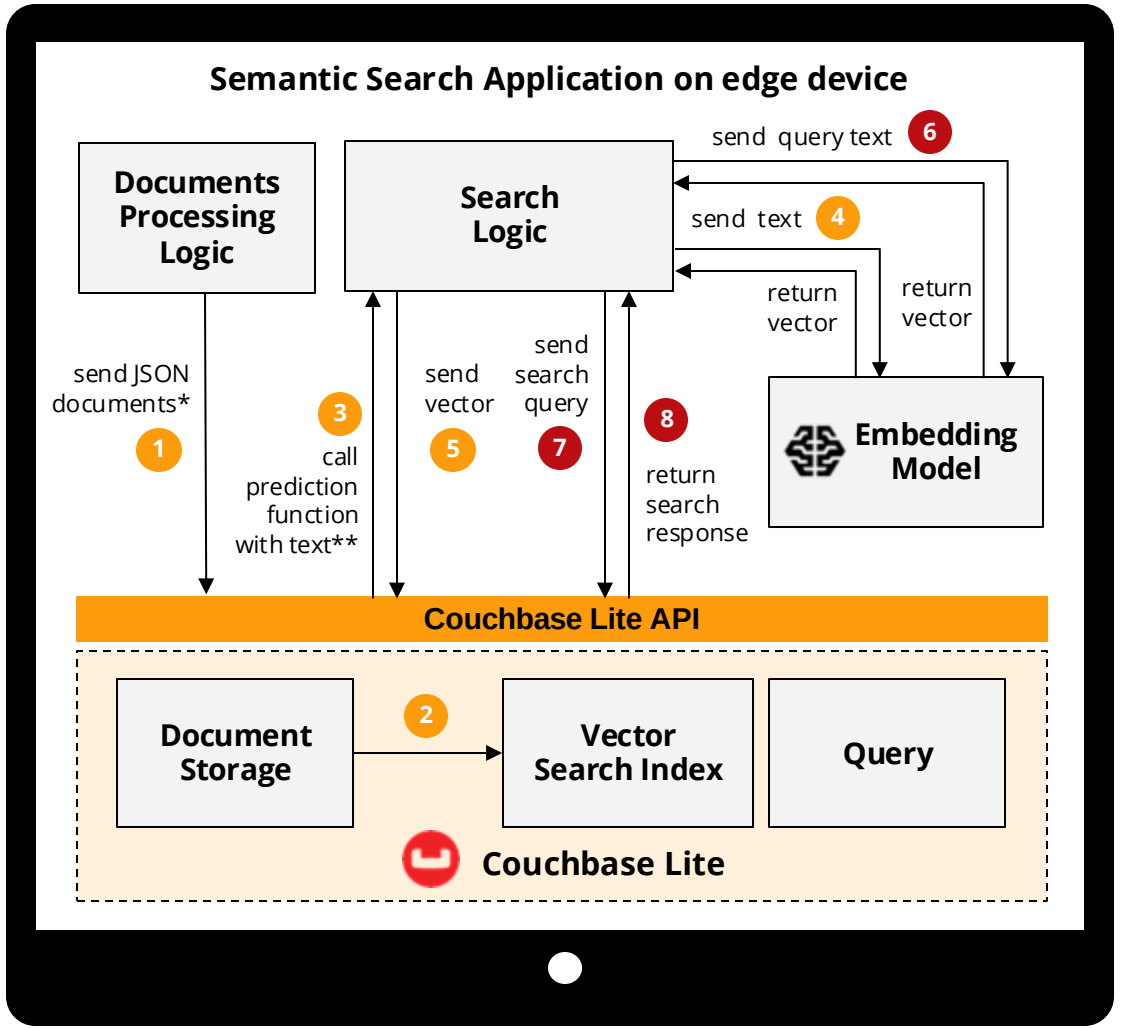
- This function accepts data input and returns vector embedding corresponding to the data input.
- Note: the function will call into a local (recommended) or remote embedding model to generate the embedding. It might not be suitable to use with the remote ML model as this can take a long time.

Vector Indexing and Process

- As documents are added to Couchbase Lite, Couchbase Lite automatically updates the vector index from embeddings generated by calling into the `predict()` function
- Automatic, synchronous process - function callback will block document update operations

Using Prediction Function to dynamically generate Embeddings

1. Application creates/updates JSON documents to Couchbase Lite. Documents do not have embeddings.
2. This triggers update to Vector Search Index on Couchbase Lite
3. Couchbase Lite calls into prediction function with text to vectorize
4. Application sends text to embedding model. Embedding model returns embedding vector.
5. Application send vector to Couchbase Lite and vector index is updated
6. Application sends "query text" to embedding model. Embedding model returns embedding Vector.
7. Application calls Couchbase Lite Service with a pure Vector search, or hybrid mixed vector search and FTS, or hybrid vector and SQL++ query.
8. Couchbase Lite returns top k-NN results of query with requested fields or objects - the vector search returns semantically similar documents



(*) Omitting index creation step for brevity (**) Can be any media

Option 2: Using Lazy Vector Indexing

Lazy Index

- Lazy Index is an alternative to using Prediction Function for generating embeddings

Full-control for Developers

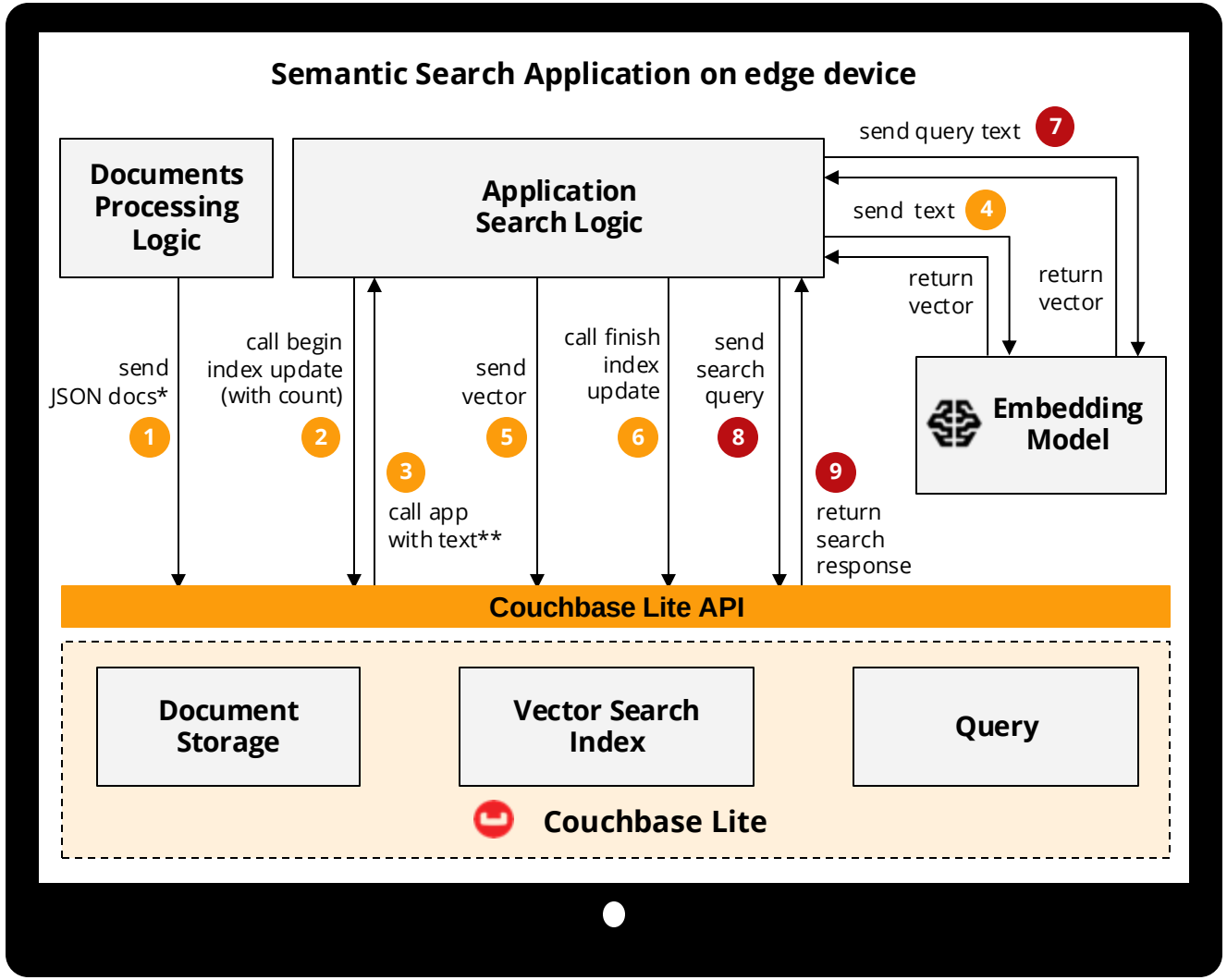
- Developers can schedule when to update the index. It is not automatic everytime document is added.
- Developers can choose the number vectors to update to the index.
- Developers can cancel or skip documents when generating embeddings.

Vector Indexing and Process

- Updating lazy index is an independent process from saving document operations
- Vector index is automatically updated when documents are deleted or purged.

Using Lazy Vector Index to dynamically generate Embeddings

1. Application creates/updates JSON documents to Couchbase Lite. Documents do not have embeddings.
2. At appropriate time, application decides to update vector Index and calls it with the count of vectors to be updated
3. Couchbase Lite calls into App with text
4. Application sends text to embedding model and gets back vector embeddings
5. Application send vector to Couchbase Lite. The vector is in memory.
6. Application commits the vector once count is reached
7. Application sends “query text” to Embedding Model. Embedding Model returns embedding Vector.
8. Application calls Couchbase Lite with a pure Vector search, or hybrid mixed vector search and FTS, or hybrid vector and SQL++ query.
9. Couchbase Lite returns top k-NN results of query with requested fields or objects - the vector search returns semantically similar documents



(*) Omitting index creation step for brevity (**) Can be any media

So should we always dynamically generate embeddings?

No! It would depend on the use case.

- Couchbase Lite는 다양한 사용 사례의 다양한 요구를 충족하는 포괄적인 솔루션을 제공합니다.
- 트레이드오프를 평가하고 사용 사례에 가장 적합한 아키텍처 옵션을 확인하세요.

Dynamically generating embeddings has implications

앱에서 구현은 더 복잡합니다.

벡터 인덱스를 생성하려면 서버 측과 Couchbase Lite 측에서 임베딩을 다시 생성해야 합니다. 특히 모델이 원격인 경우 매우 느린 프로세스가 될 수 있습니다.

아직 색인되지 않은 문서는 검색에 참여할 수 없습니다.

에지에 임베딩 모델이 없는 경우 클라우드에서 임베딩을 생성해야 합니다.

Visual Search:
Image Recognition with Vector Search in Couchbase Mobile

What do we mean by “visual search”?

See it

Zero in on a specific item

Identify it

Understand what you see

Respond

React based on what you see

Retail point-of-sale example | The Problem

Internet-connected self checkout kiosk

Manual search for items is error prone

Barcodes are imperfect

Internet dependencies slow transactions

Business stalls when connection fails

Convenient, but….

- Hunt-n-peck search means mistakes and hitting “back” button
- Barcode labels can wrinkle, fall off, and be used fraudulently leading to **retail shrink**.
- Network latency and slowness cause double scans and **mistakes**
- Slower for each transaction, downtime w/o internet = **unhappy customers**
- Fewer transactions per day = **LOST $$ for the grocer**

Retail point-of-sale example | The Solution

3 seconds per item (+or-)

“Smart-cart” self checkout

Cart-fixed tablet based scanners

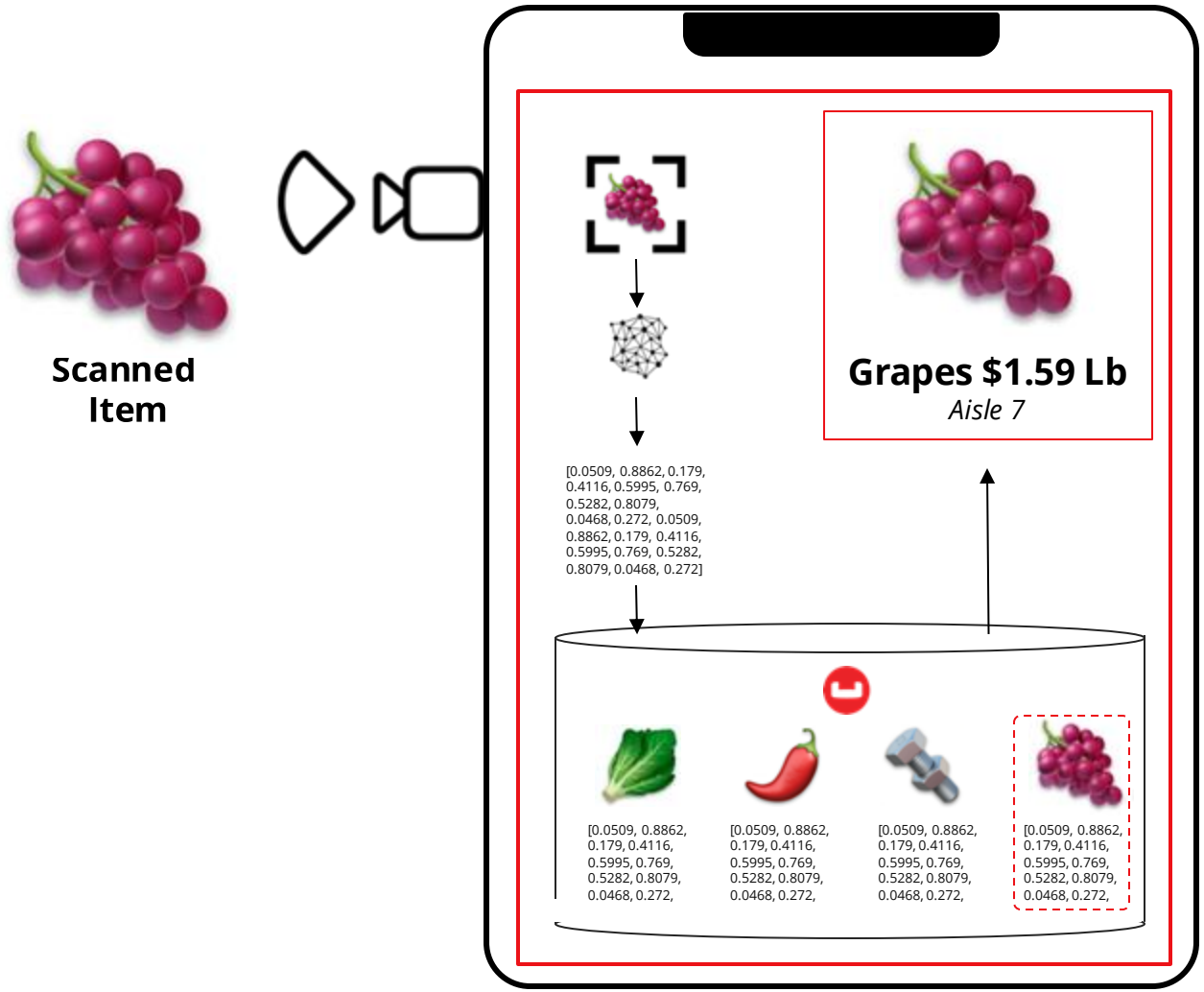
Embedded data processing and local AI models

Speed transactions with guaranteed uptime

…easier, faster, more accurate, and truly convenient

- Hold item in front of camera, confirm, place in cart
- Works with or without connectivity
- Faster and easier shopping experience = **happy customers**
- More transactions + less shrink = **MORE $$ for the grocer**

Couchbase Lite | Vector Search For Image Lookup



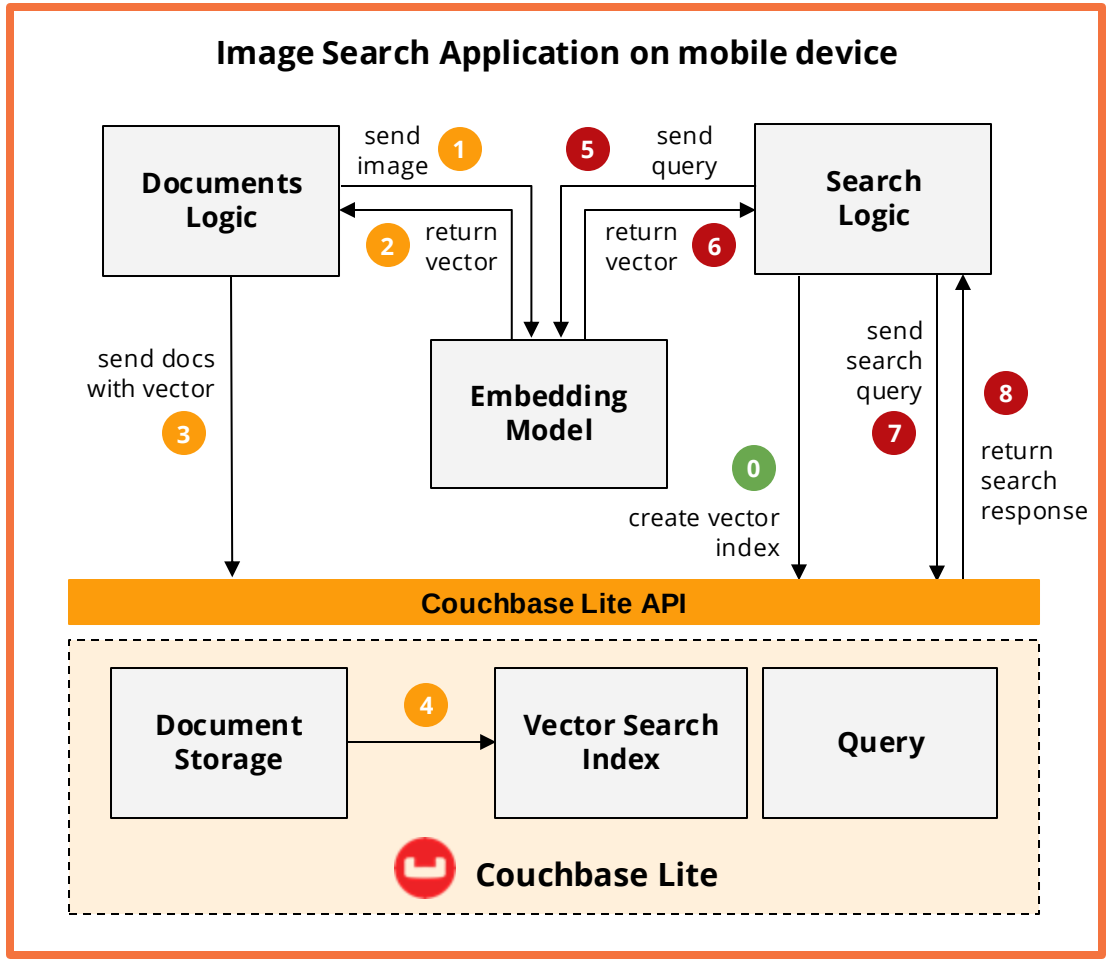
Tablet-based Point-of-Sale app

- Item is scanned with device camera
- Vectorized with local embedding model
- Couchbase Lite vector store is searched
- Nearest vector match found
- App displays matching item and data

In More Depth | Image Search with Couchbase Lite

Image Search Application running on edge device with embedded Couchbase Lite

0. Application creates vector index
1. Application sends documents to embedding model
2. Embedding Model returns embedding vector and application adds vectors to JSON documents
3. Application creates/updates JSON documents (image with vector) to Couchbase Lite
4. This triggers update to vector search index on Couchbase Lite that updates index with document
5. Application sends captured image to Embedding Model
6. Embedding Model returns corresponding embedding vector
7. Application calls Couchbase Lite with a pure vector search, or hybrid mixed vector search and FTS, or hybrid vector and SQL++ query
8. Couchbase Lite returns top k-NN results of query with requested fields or objects - the vector search returns similar images



(*) can be any media

Use cases for visual search at the edge with Security

Purchase Items

Scan and purchase goods using a mobile app

Price Lookup

In-store kiosks for price and related information for goods

Insurance Claims

Assess images of damage and recommend replacement parts and service

Self Check In

Scan a pass or use facial recognition for instant check-in

Healthcare My Data

Provide differentiated services in the health care sector and utilize them in various health care service fields

Smart Mirror

In-store mirror device shows user in different outfits and accessories

Smart City

Use camera images to assess street conditions and recommend services

Security Surveillance

Visually detect and identify possible security issues, find missing persons, etc.

Defense

Cameras on drones enable autonomous threat detection

Game

Gamers expect to play whenever they want to for as long as they want to
A low tolerance for noticeable delays, lag being one of them

Hands-on Lab

> Capella 기본 구성

3시 정각에 다시 시작하겠습니다.

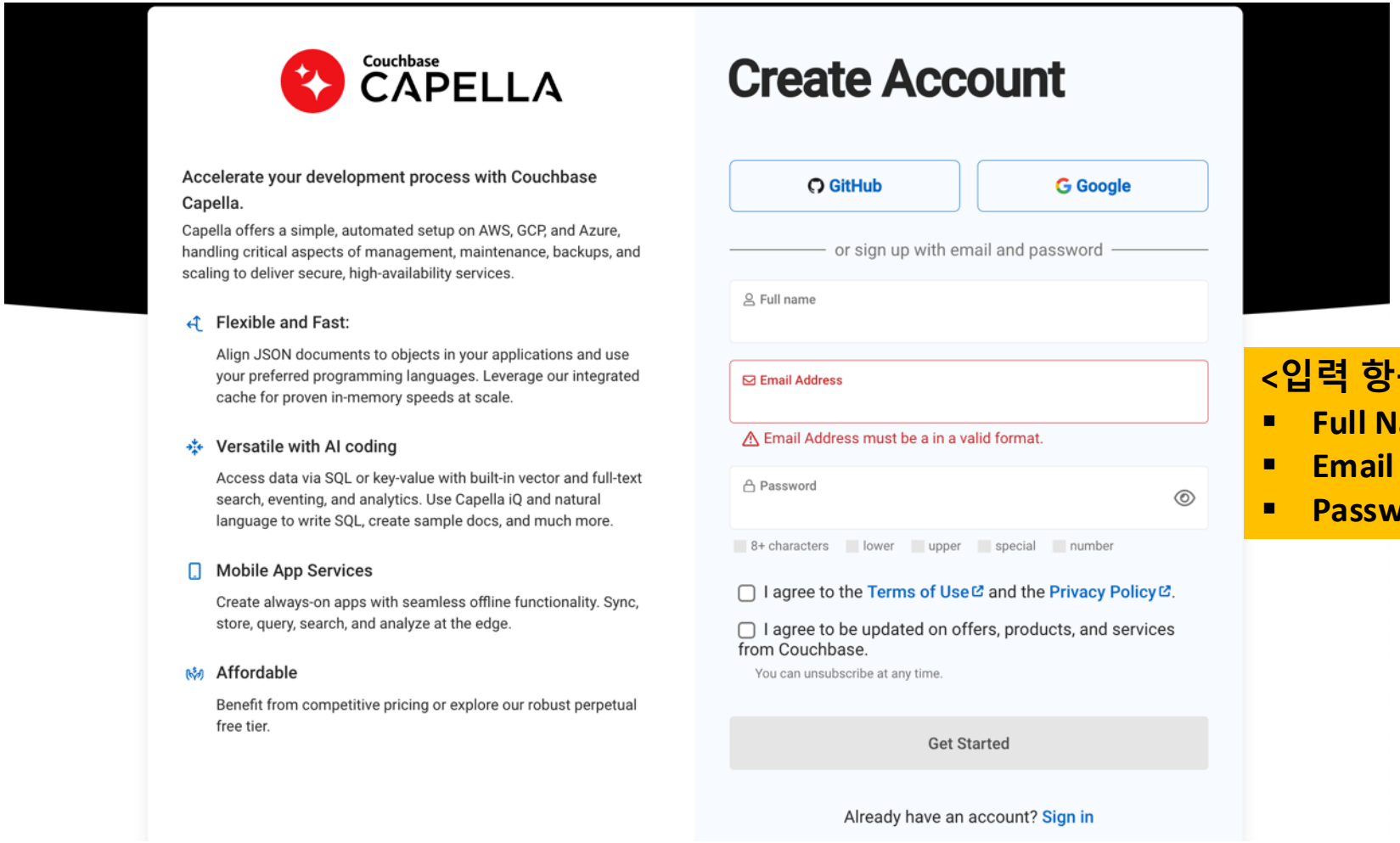
질문 : 챗팅 방에 남겨 주시면 감사하겠습니다.

>

Capella
free-tier
Operational DB 구성

Couchbase Capella Sign-up

Sign-up URL : https://cloud.couchbase.com/sign-up

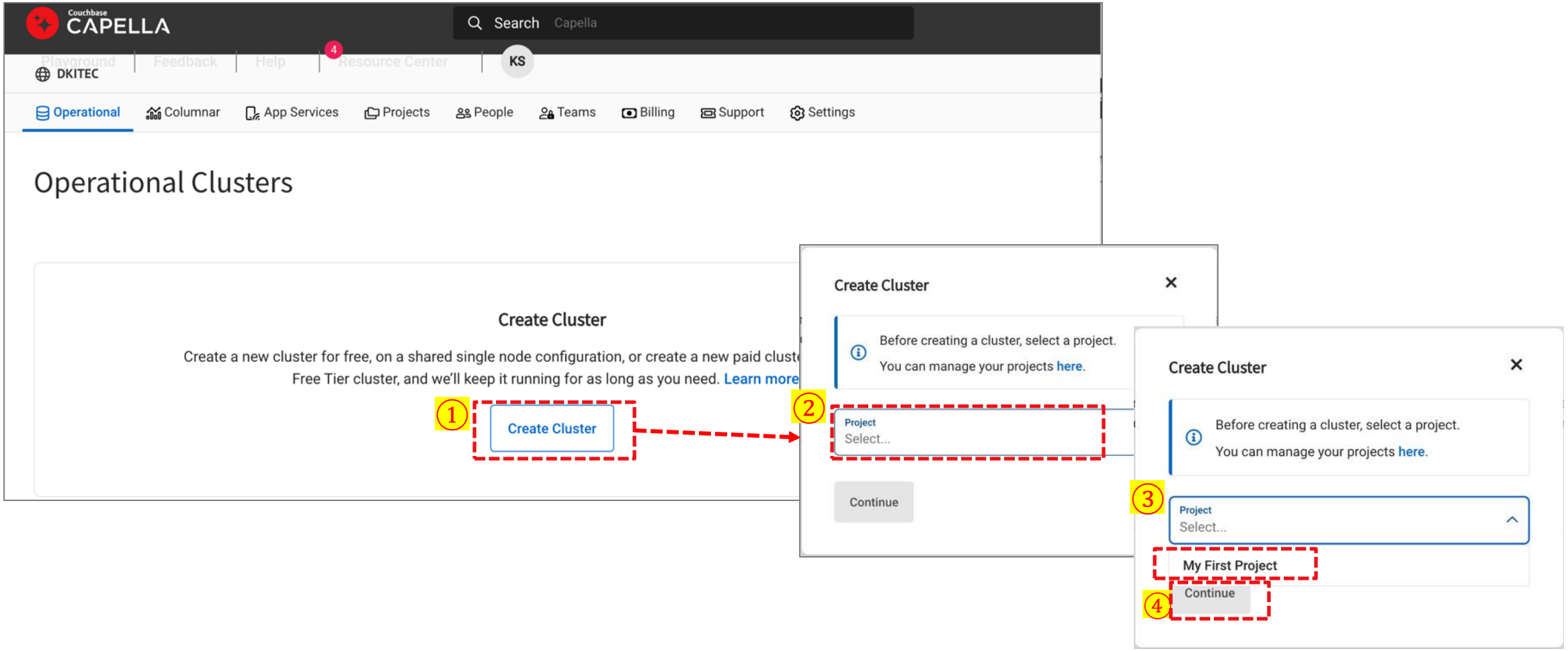


<입력 항목>

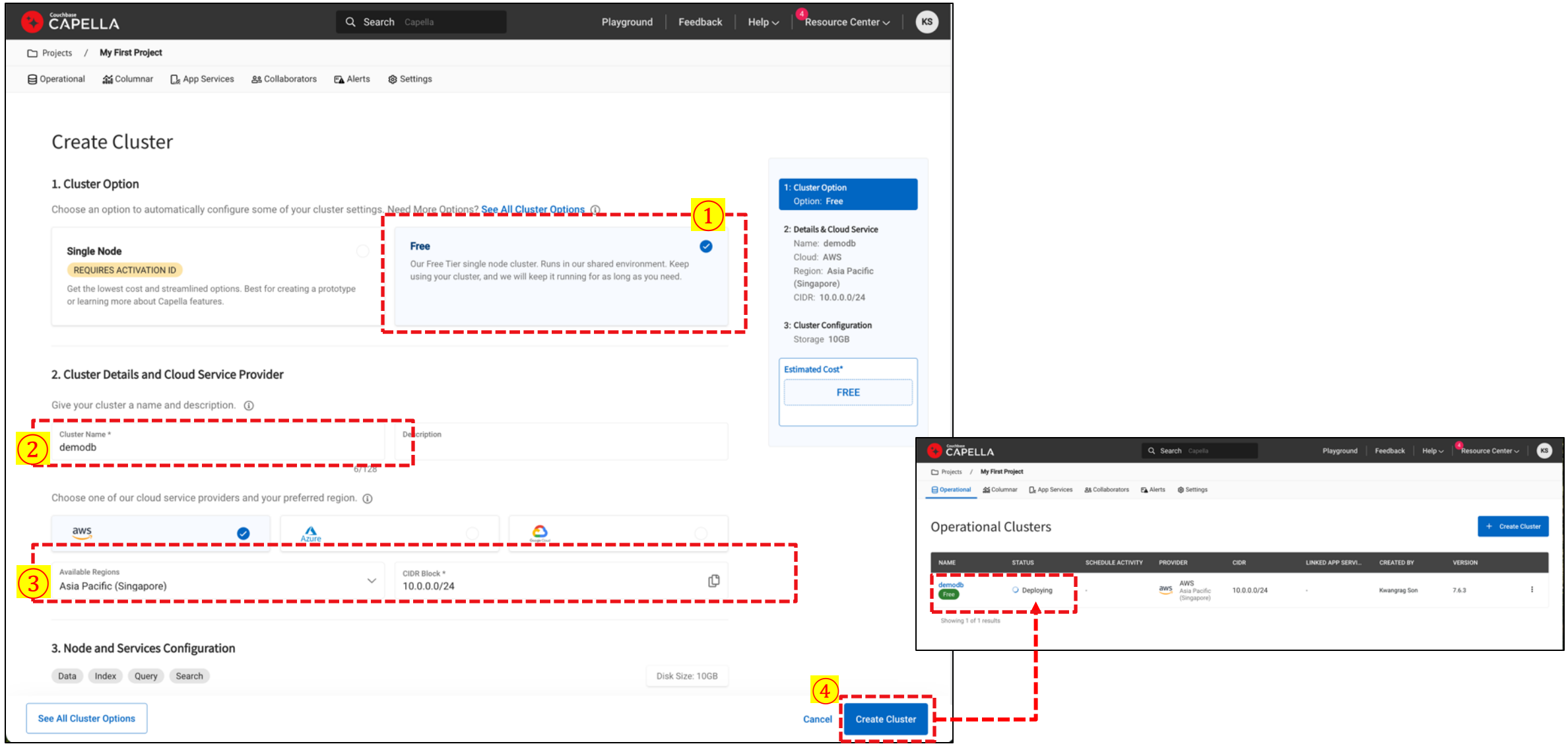
- **Full Name**
- **Email : 계정 및 확인 메일 전달**
- **Password**

Capella > Database(Operational Cluster) 생성

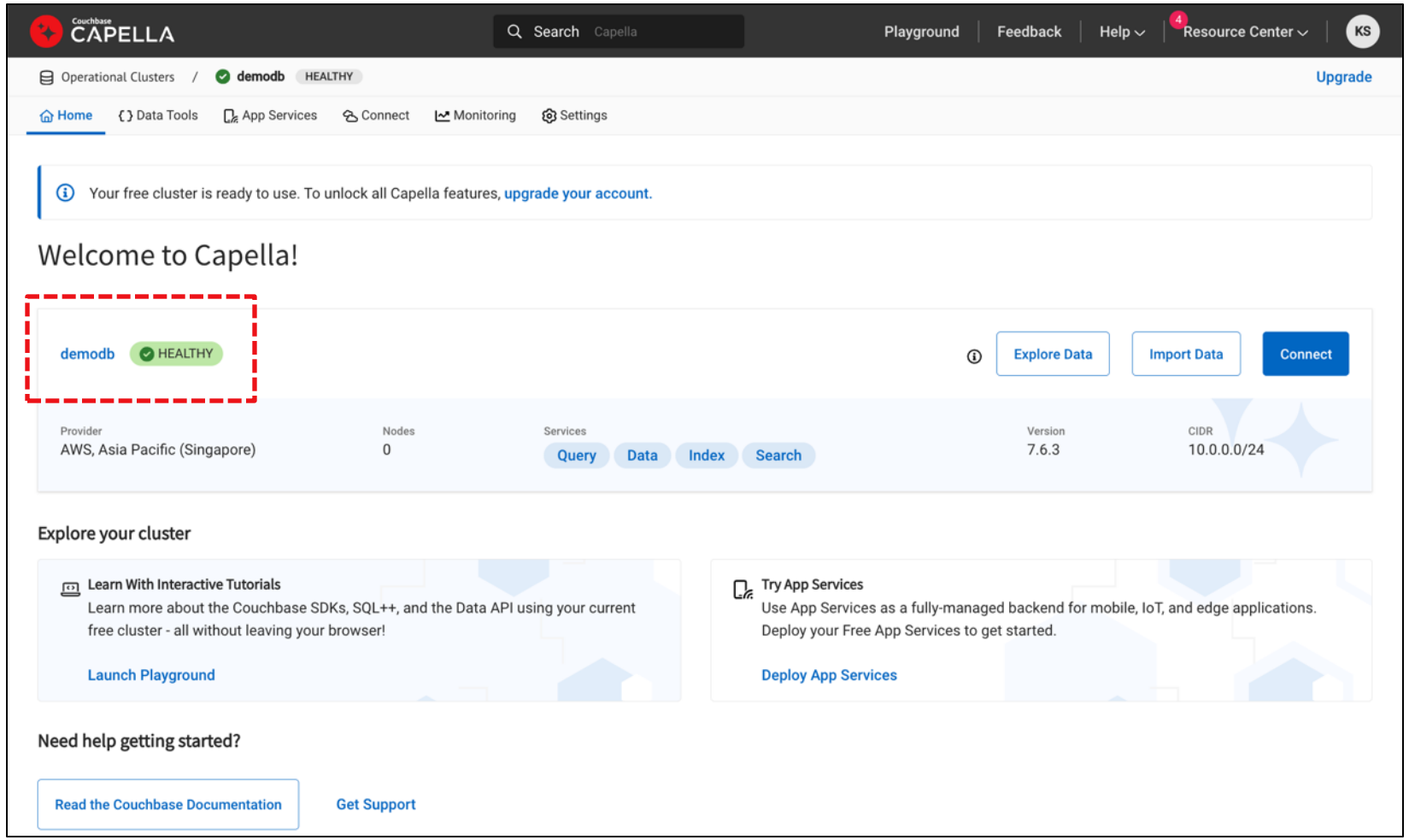
기본 메뉴 설명 : Project > Databases, App Services(Sync Gateway)



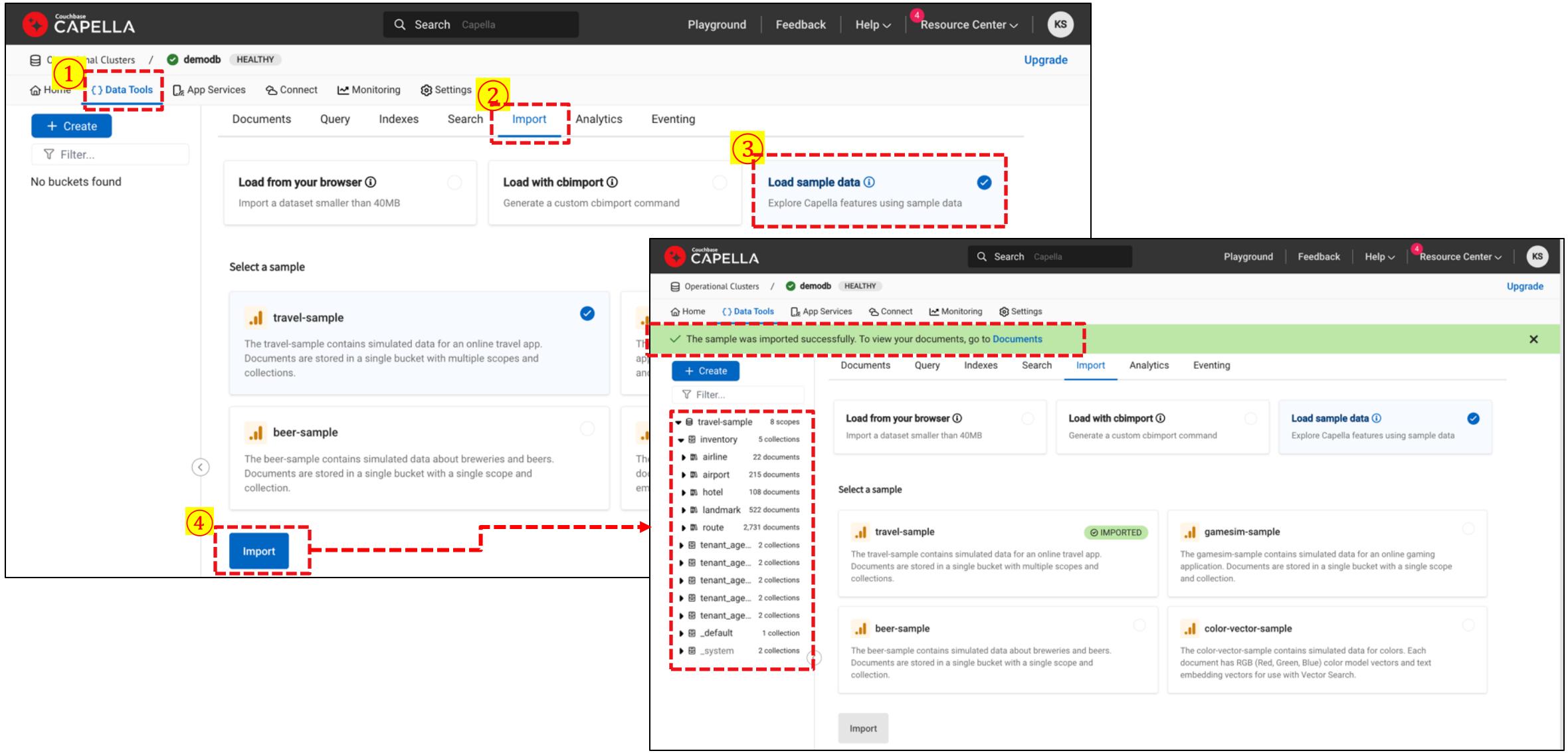
Capella > Database(Operational Cluster) 생성



Capella > Database(Operational Cluster) 생성 완료

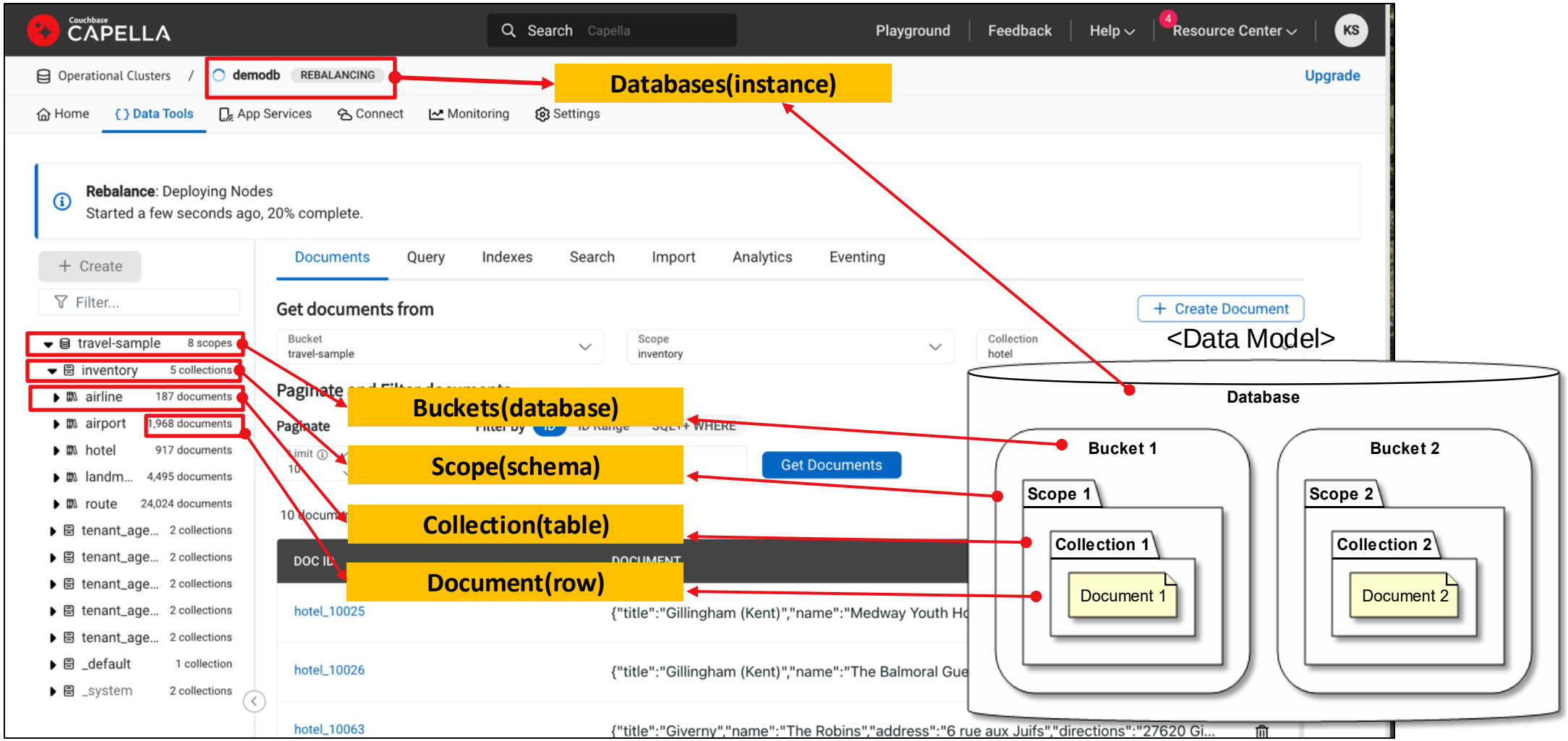


Capella > Database(Operational Cluster) > 샘플 데이터 로딩



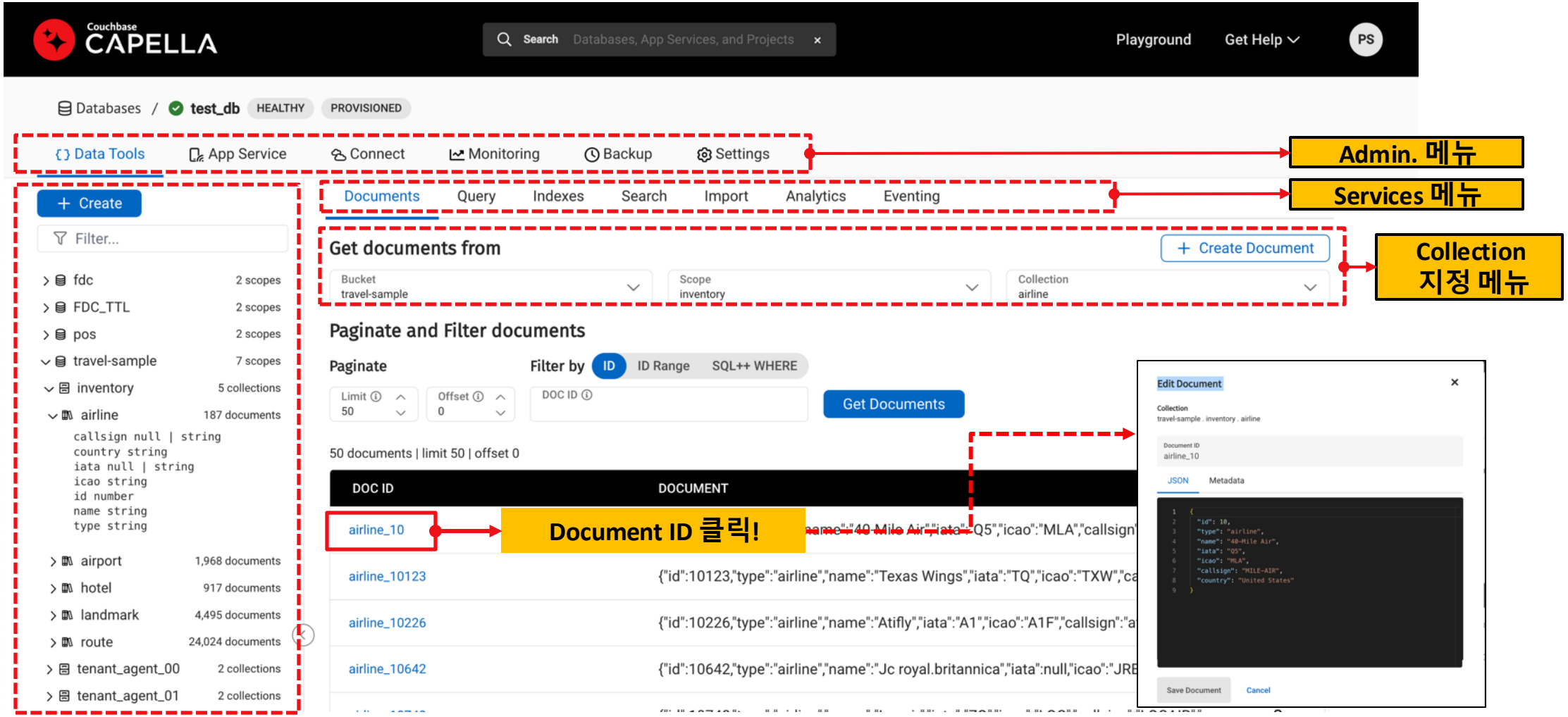
Capella UI

기본 메뉴 > Databases > Bucket : Data Tools

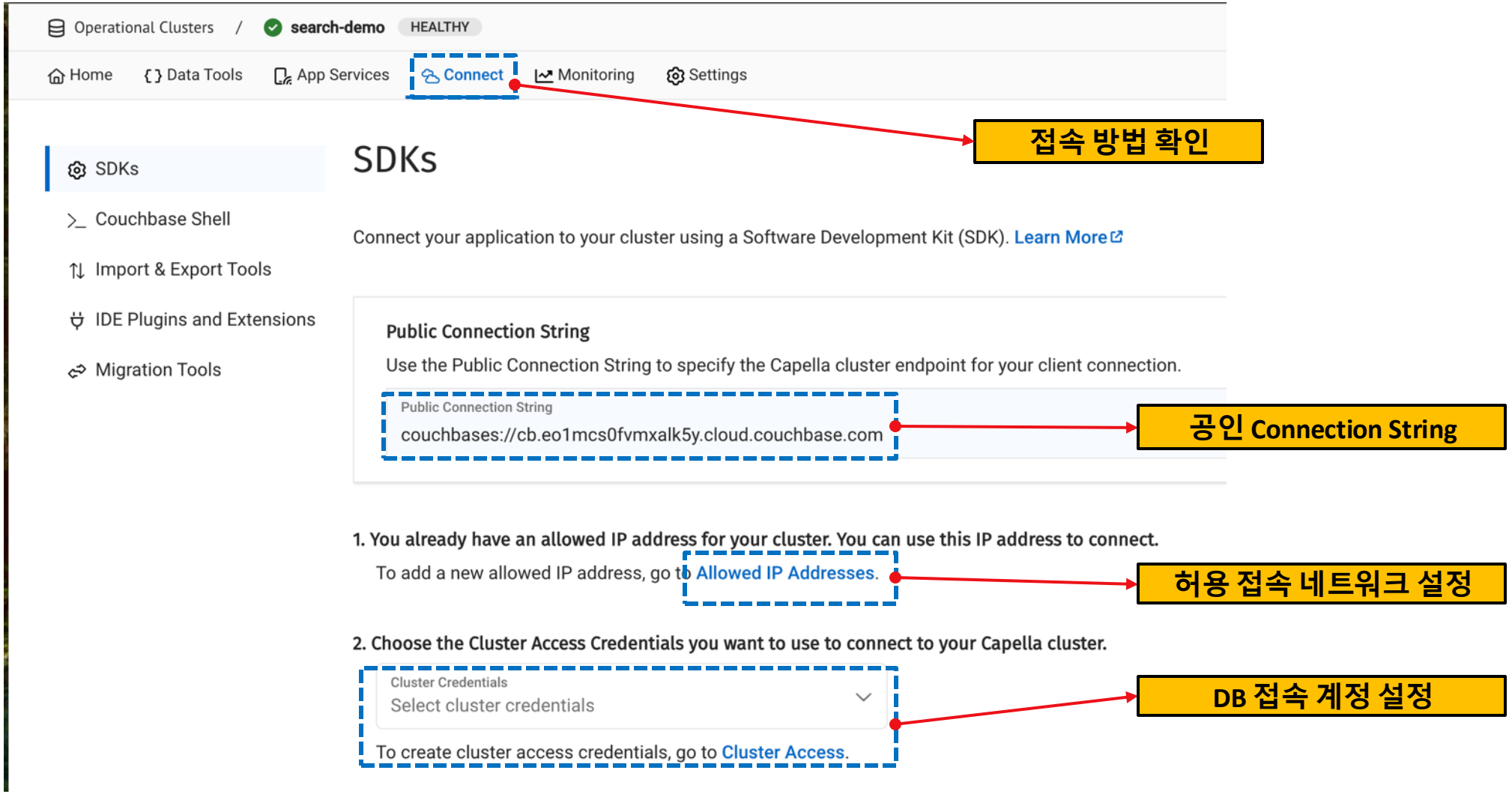


Capella UI

기본 메뉴 > Databases > Bucket : Data Tools > Documents

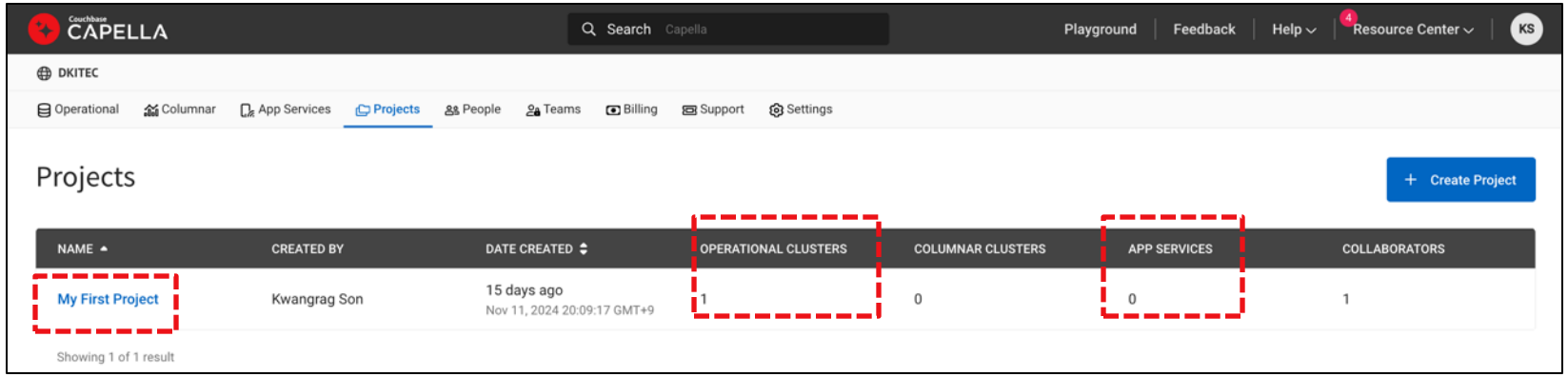


Capella > Operational Cluster 연결



Capella > 기본 메뉴 설명

기본 메뉴 설명 : Project > Databases, App Services(Sync Gateway)

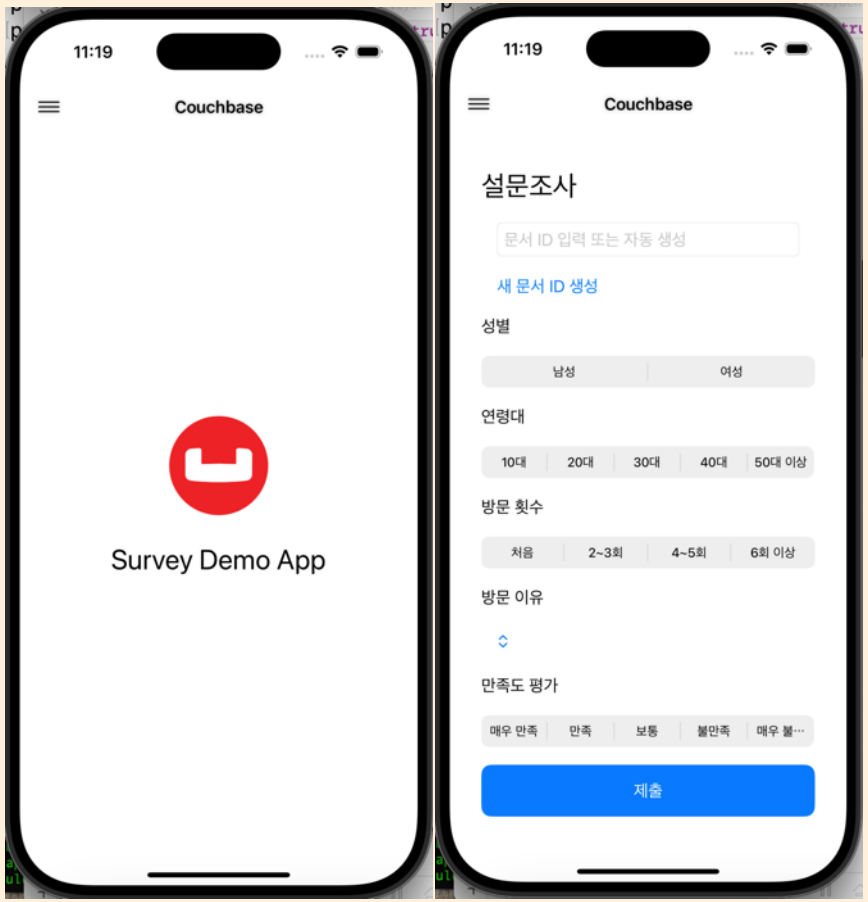


- App Service 생성 및 구성은 52페이지 이하 참고

Capella
free-tier
App Service 구성

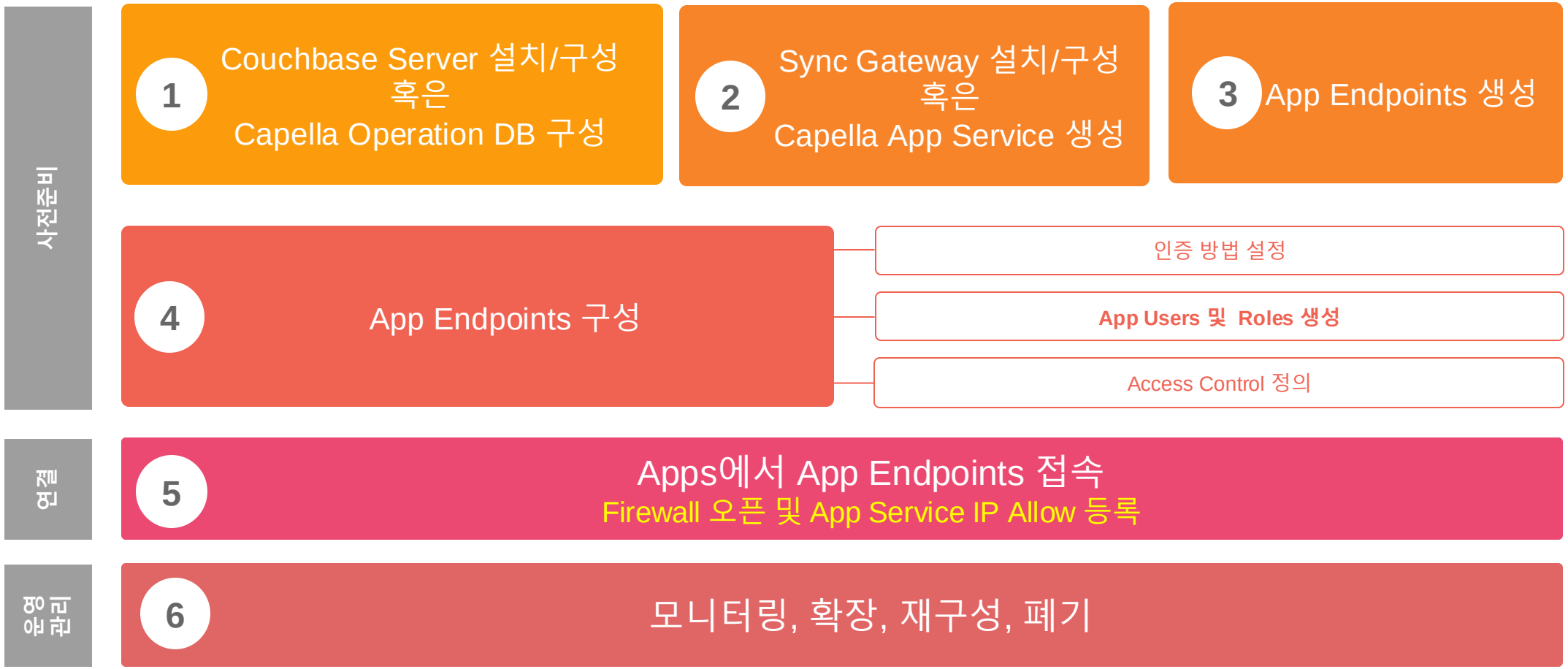
Agenda

1. Couchbase Mobile 환경
2. Capella APP Service 구성
3. Survey Demo App 실행

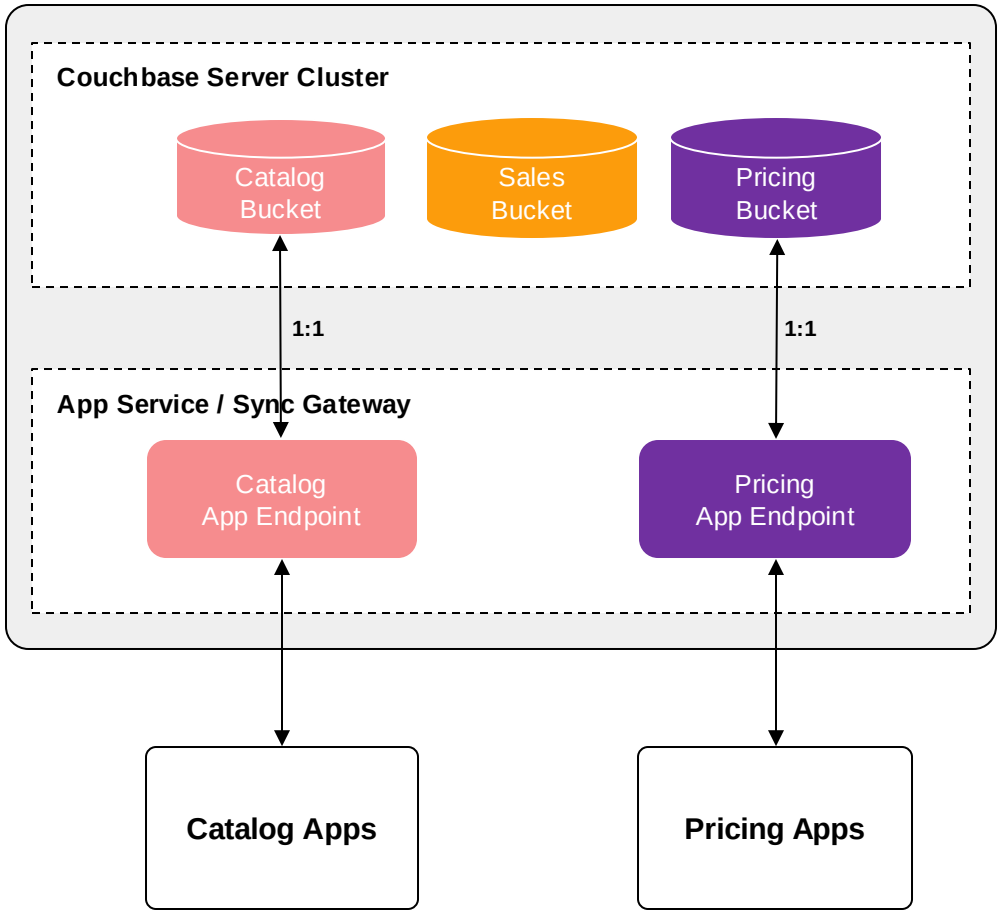


1-1. Couchbase Mobile 환경 > 구성 절차

구성 1안(설치형) : Couchbase Server ⇔ Sync Gateway ⇔ Couchbase Lite
구성 2안(관리형) : Capella (Operational DB ⇔ App Service) ⇔ Couchbase Lite

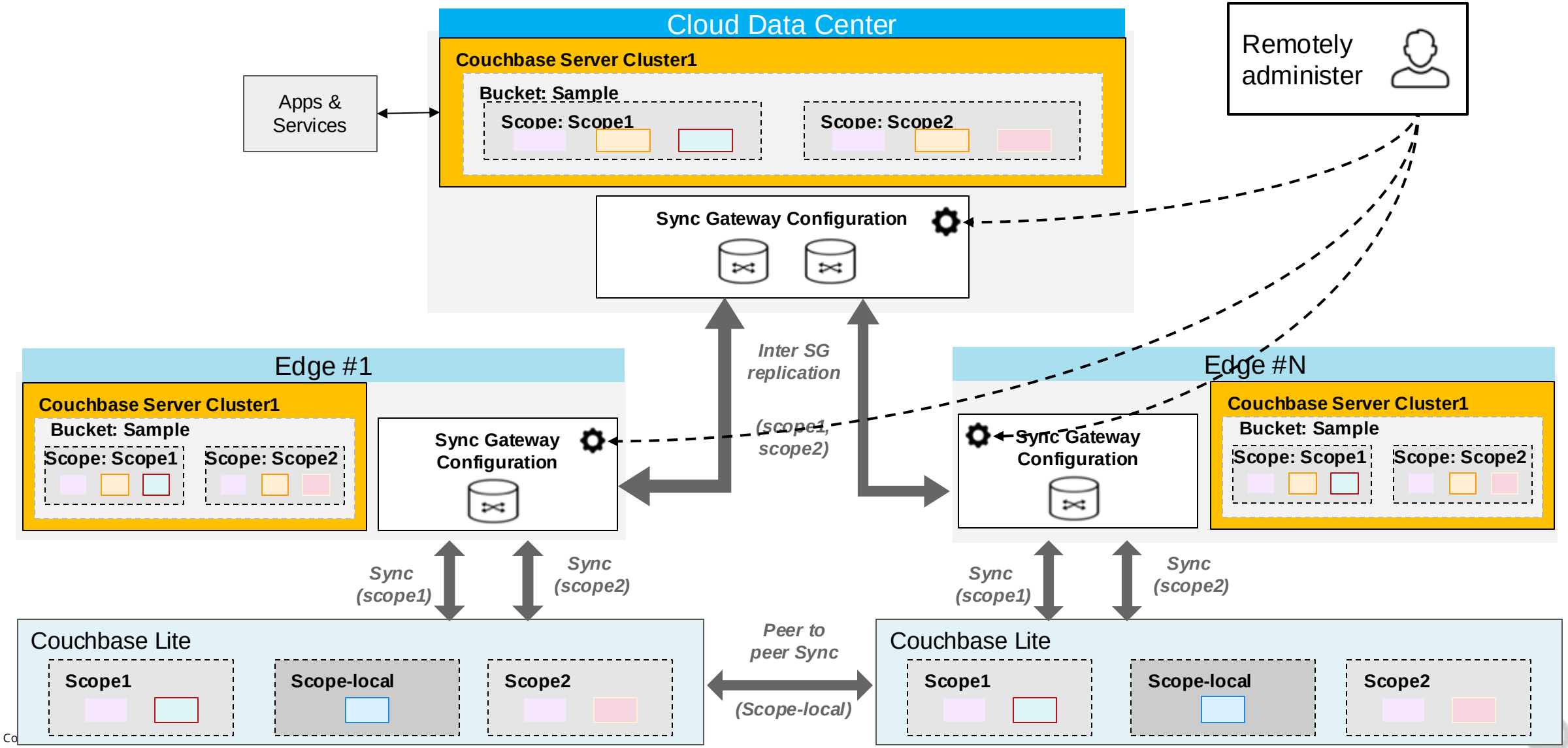


1-2. Couchbase Mobile 환경 > App Endpoints

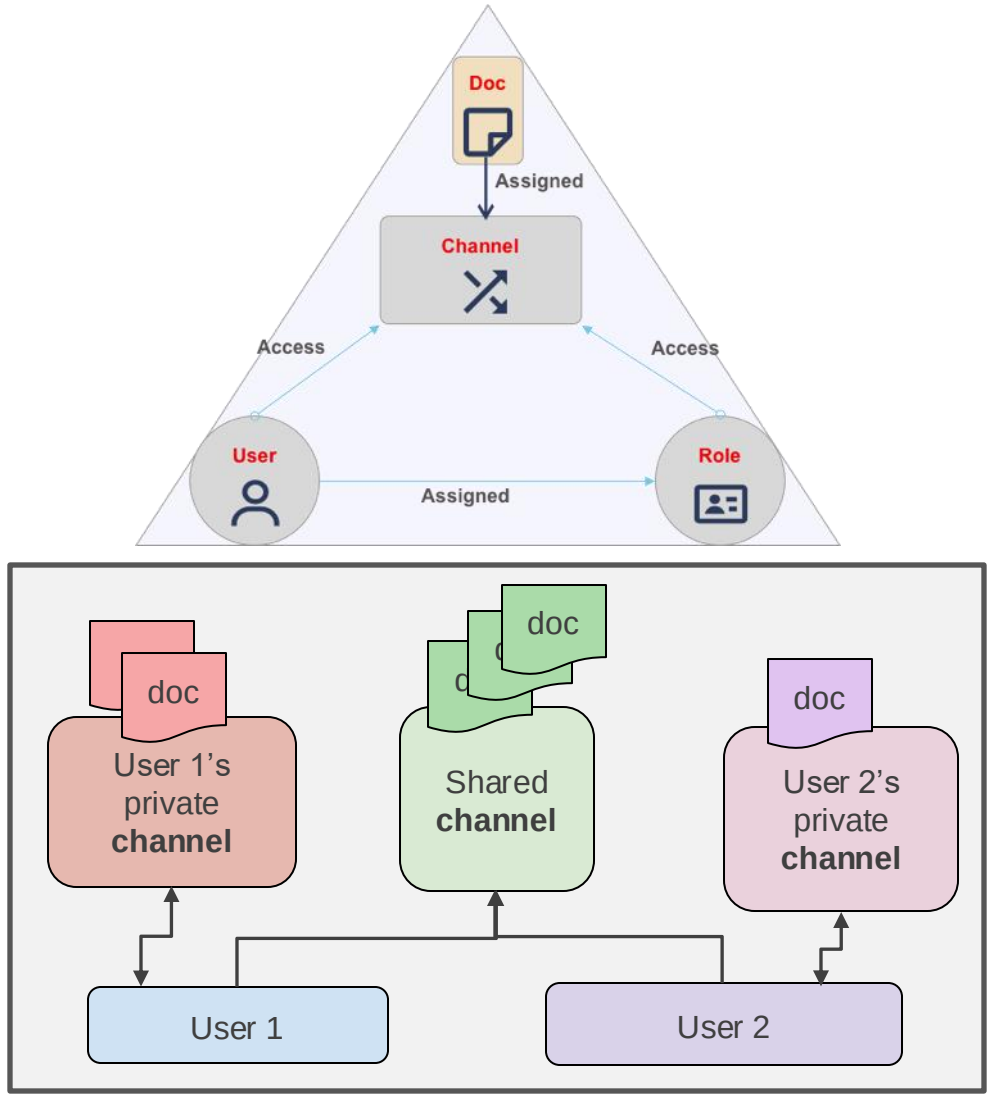


- 특징
  - App에서 App Service를 접근할 수 있는 경로 제공
  - 1 개의 Bucket과 연결 설정 가능
- 참고 내용
  - 연결된 Bucket에 "Sync Gateway DB"가 생성되고 해당 Bucket 내에 데이터 저장
  - 구성 정보는 모든 App Service 노드에 적용

참고 > 논리 레퍼런스 아키텍처



참고 > Access Control 아키텍처

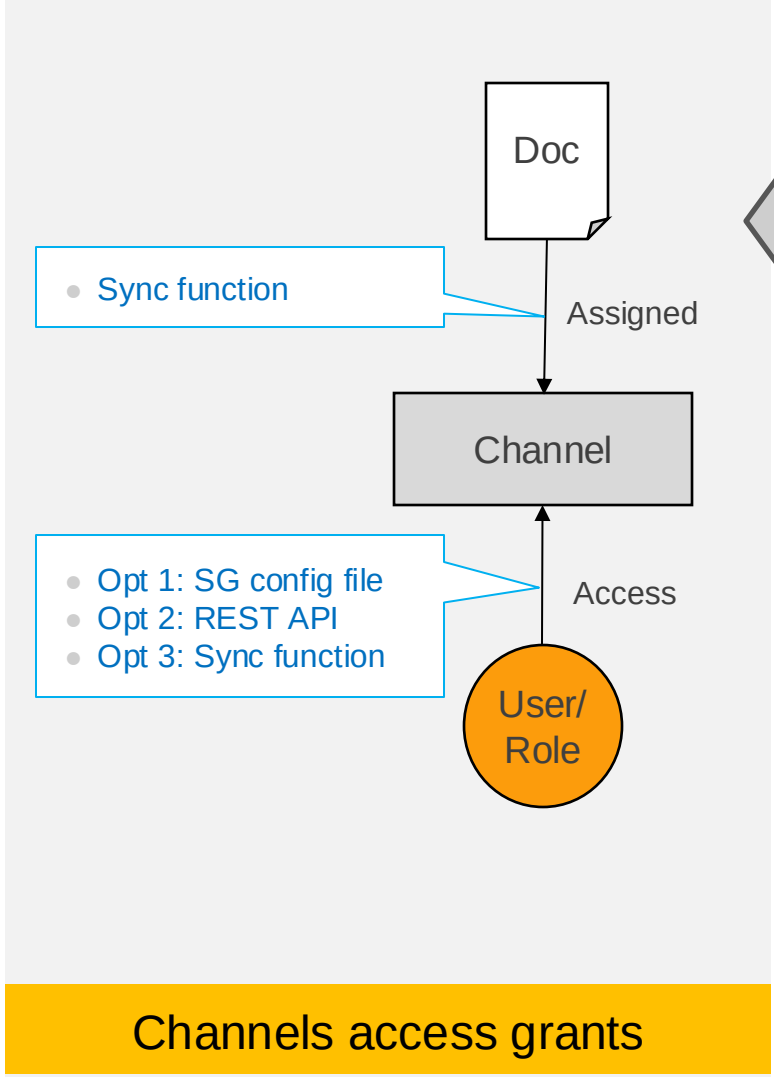


- 문서를 채널로 라우팅하고 해당 유형의 문서에 액세스할 수 있도록 하려는 사용자 또는 역할이 해당 채널에 액세스할 수 있도록 하여 문서 액세스를 제어
  > 접근 제어 구성 (채널, 사용자, 롤)

- 모든 다큐먼트를 1개 이상의 채널에 할당
- User에게 채널에 접근하기 위한 권한 부여
- Role은 유사한 접속 권한을 가진 User 그룹
  - Role에게 채널에 접근할 권한 부여
  - User는 특정 Role에 특정 등록

참고 : https://docs.couchbase.com/sync-gateway/current/access-control-concepts.html

참고 > Access Control 적용



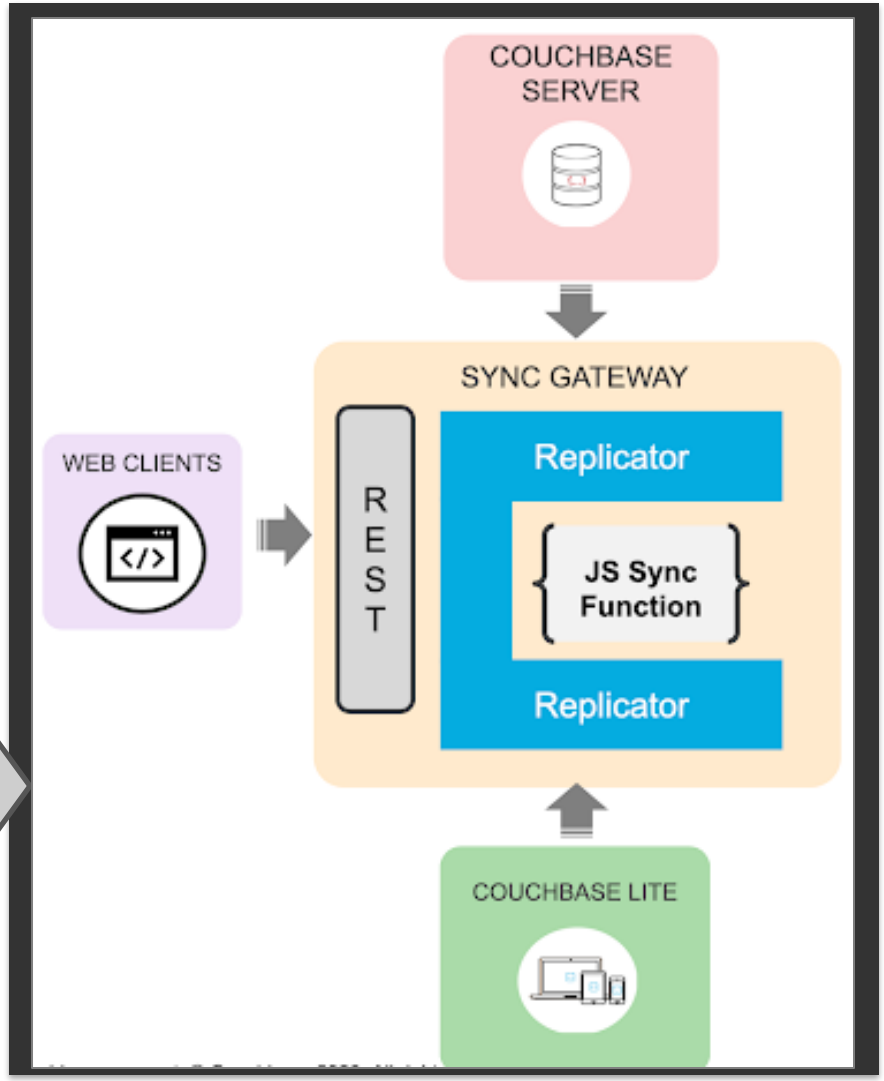
Channels access grants

적용 방식

- Document를 Channel에 할당
  - Sync 함수
- User에 특정 Channel 접근 권한 부여
  - Configuration File
  - REST API
  - Sync 함수

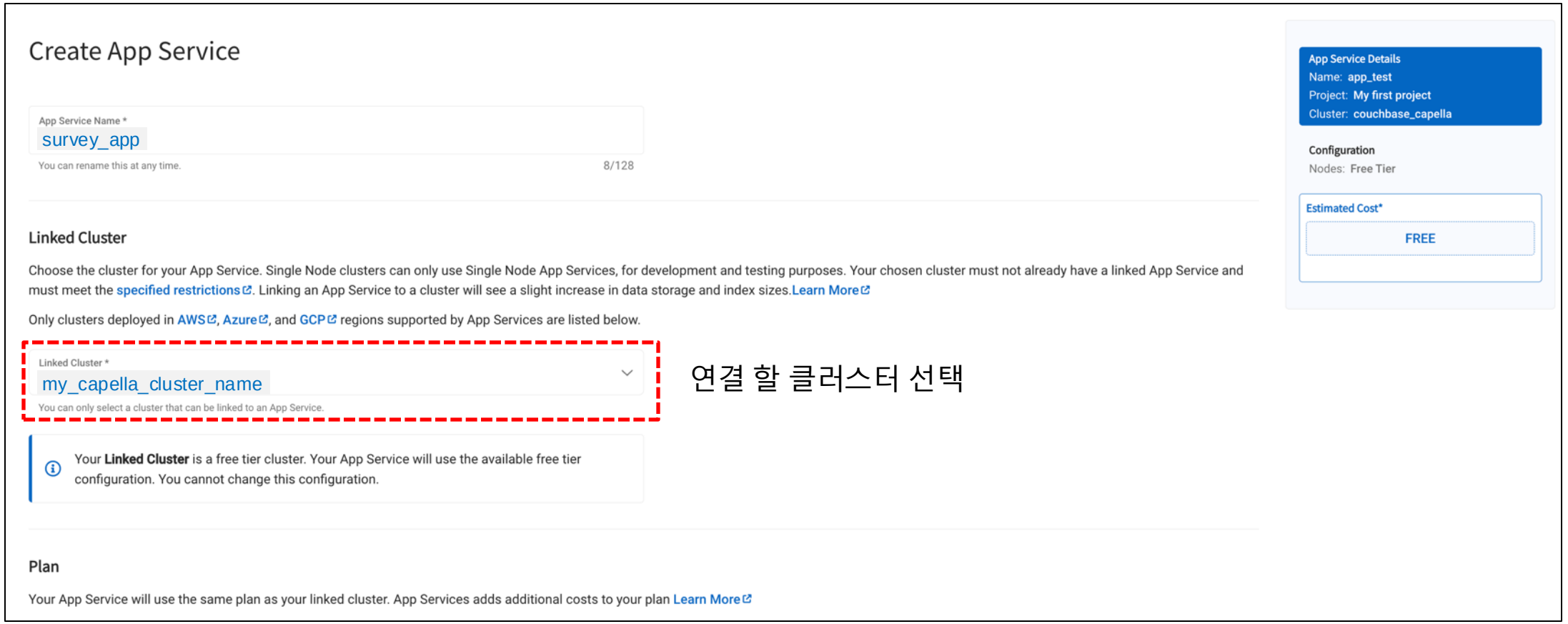
Sync 함수

- Configuration File에 정의하는 JavaScript 함수
- Document가 변경되는 시점에 항상 호출
- 기능
  - Authorization
  - Validation
  - Access Grant
  - Channel Assignment



2-1. Capella App Service 생성

- Capella App Service 생성 : survey_app

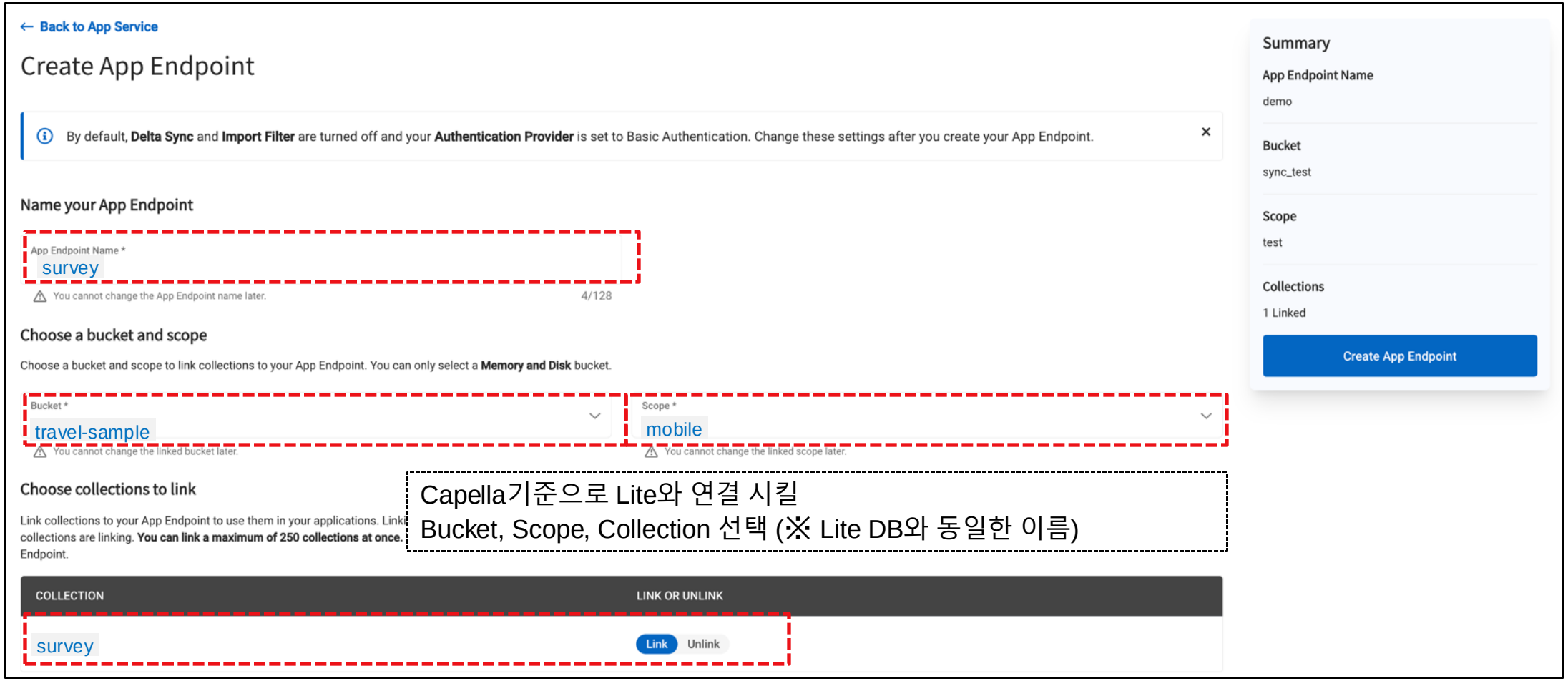


연결 할 클러스터 선택

※ App Service 생성 전 Lite 동일한 이름의 Scope, Collection 생성

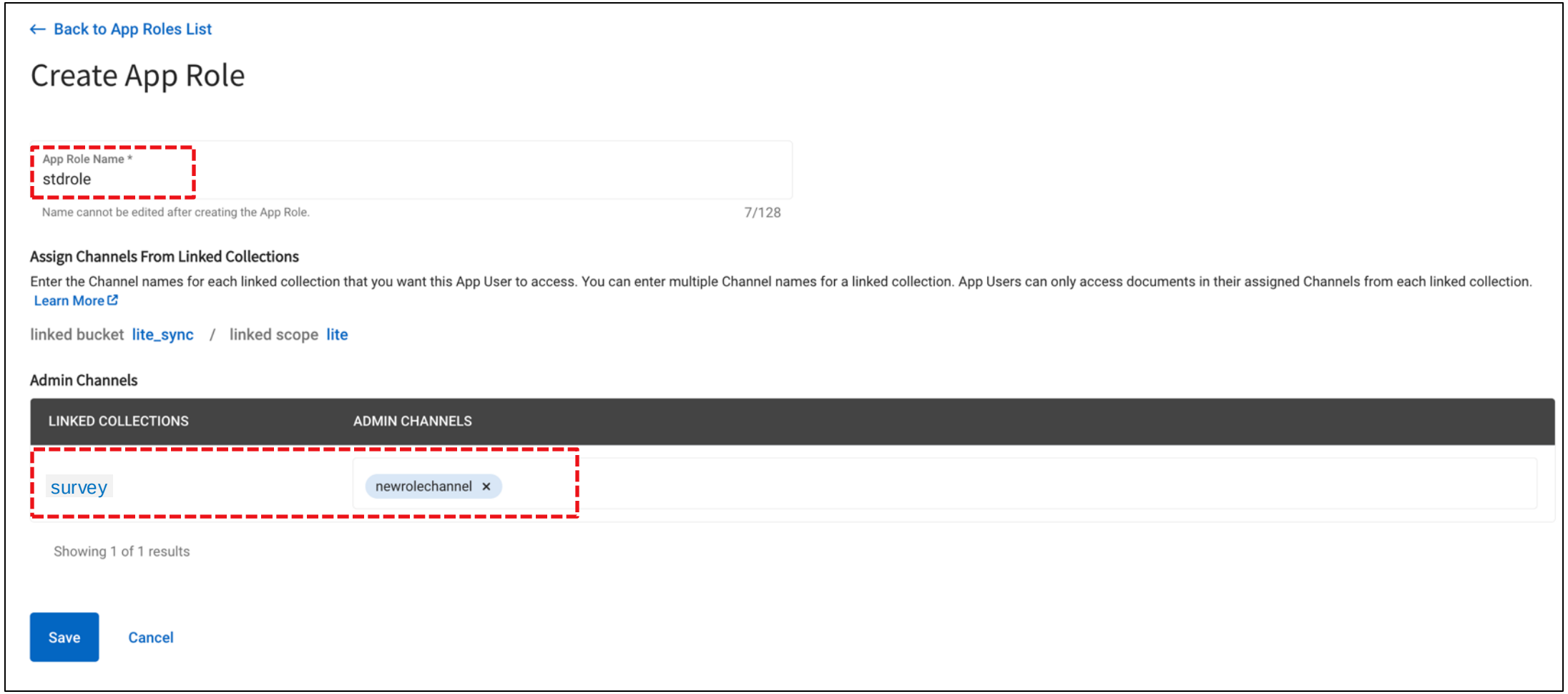
2-2. Capella App Service > Endpoint 생성

- Endpoint 생성 (Endpoint : survey, Bucket : travel-sample, Scope : mobile, Collection : survey)



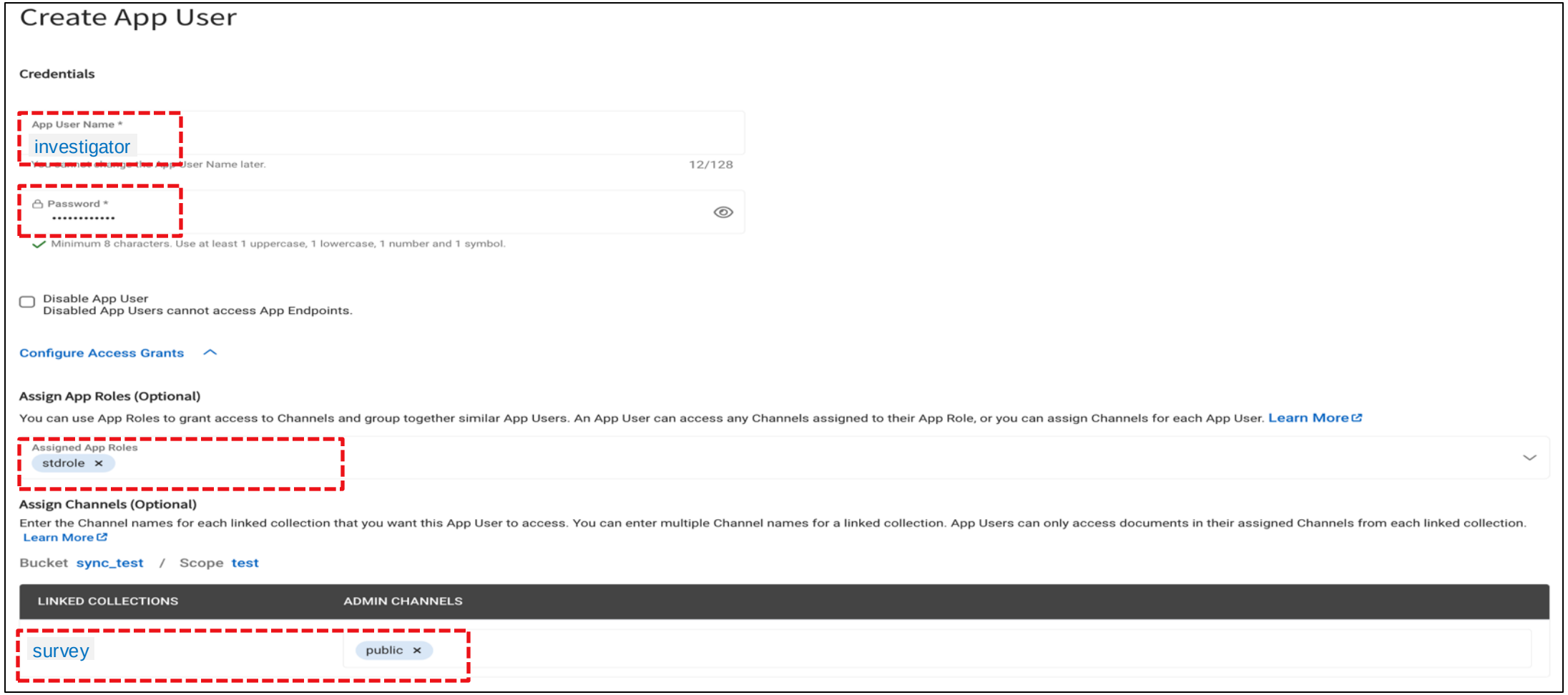
2-2. Capella App Service > Endpoint > Role 생성

- Role 생성



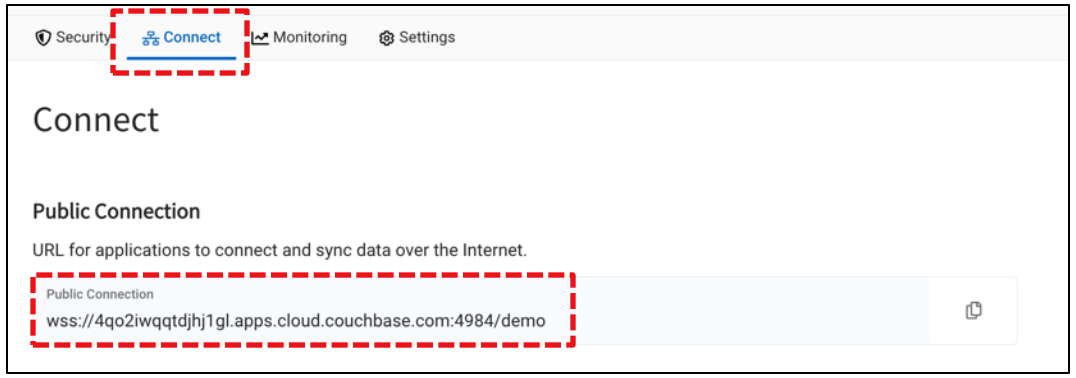
2-2. Capella App Service > Endpoint > User 생성

- Endpoint에 접속할 User 생성 : Username : investigator , password : Passw0rd!

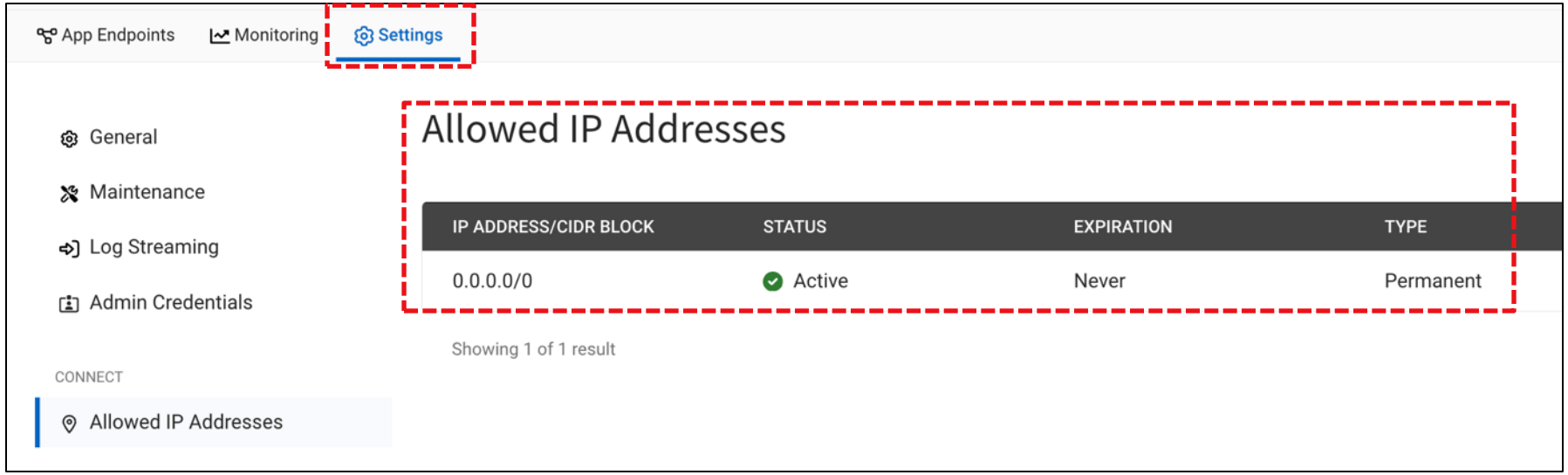


2-3. Capella App Service > Allowed IP Addressess 등록

- AppService IP and 접속 허용

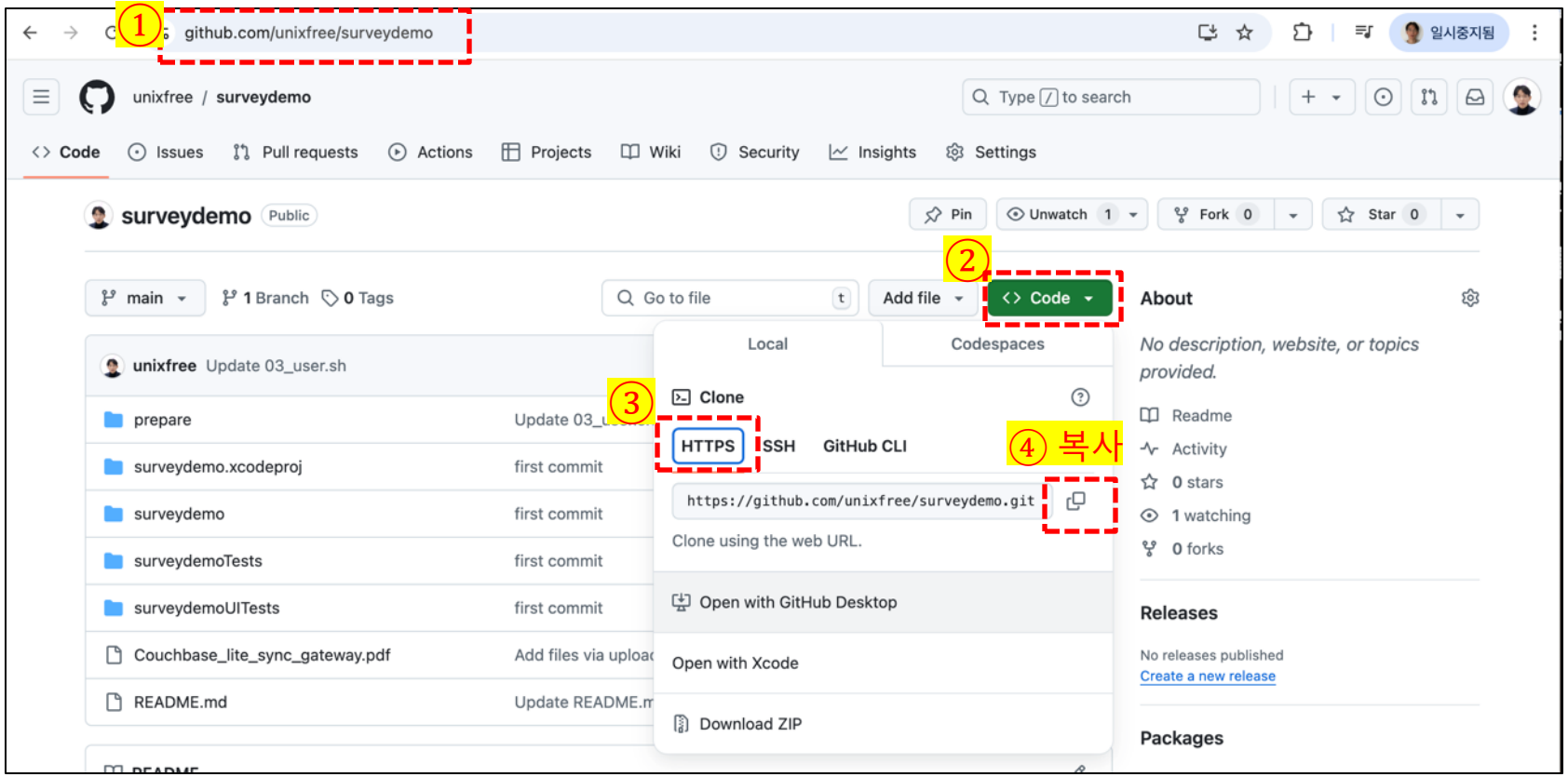


1. AppService 연결시 주소창의 IP가 아닌 Connect Tab
   > Connection IP 필요
2. Settings Tab -> IP 접속 허용 필요



3-1. App 소스 다운로드

- Survey 데모 앱 소스 다운로드
  - https://github.com/unixfree/surveydemo

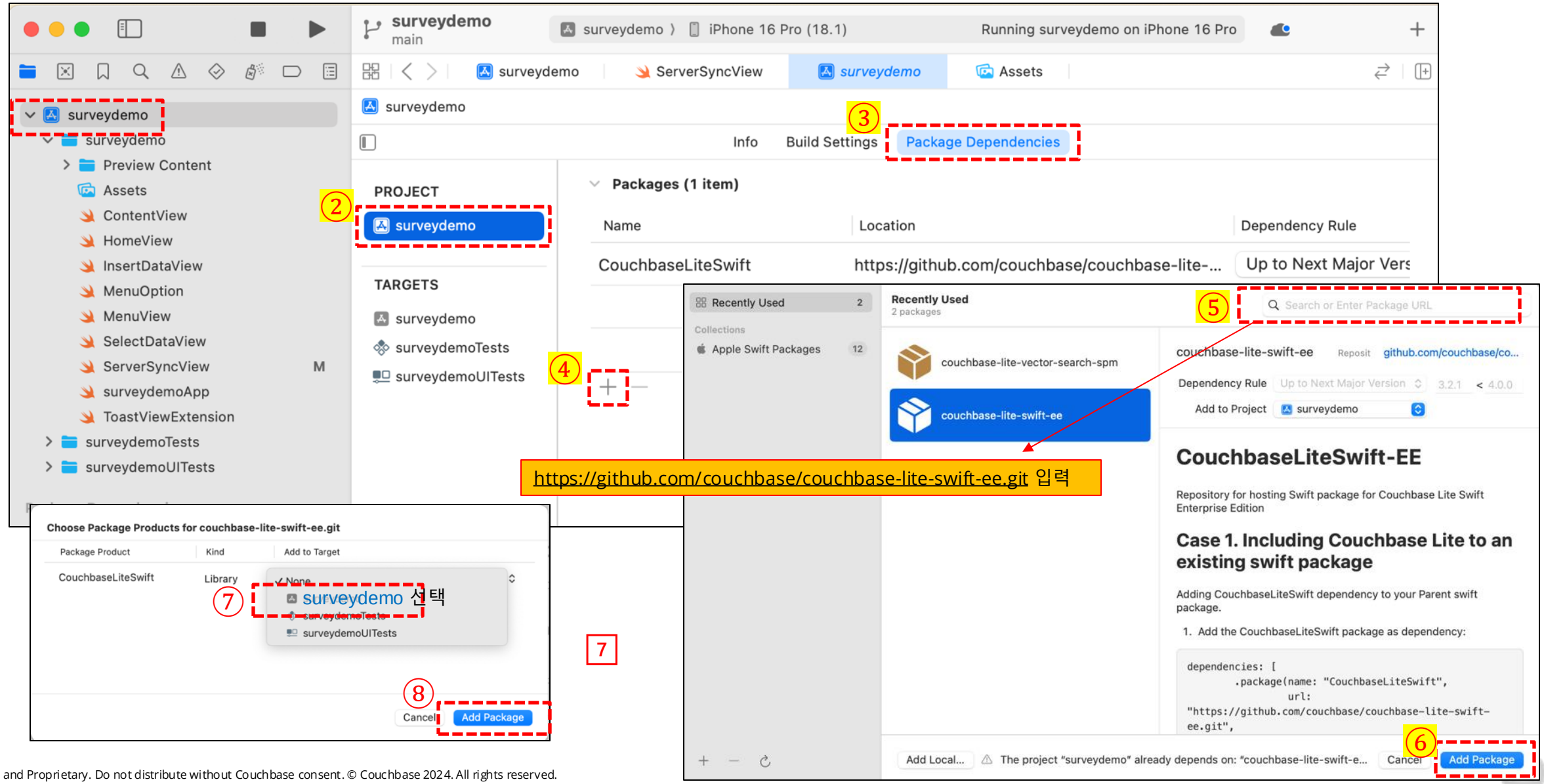


⑤

```
$ cd my_folder
$ git clone https://github.com/unixfree/surveydemo.git
```

⑥ Xcode 실행 후, File > Open… > **my_folder** > **surveydemo** > **surveydemo.xcodeproj** 선택 후 > **Open** 클릭

3-2. Xcode에서 Couchbase Packages 설치



3-3. App Connect Code 설명

- InsertDataView.swift

```swift
// 데이터 제출 함수
func submitData() {
    // ID 유효성 검사: 입력된 ID가 비어 있을 경우 경고 표시
    guard !documentID.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty else {
        showIDWarning = true
        return
    }

    showIDWarning = false // ID가 비어 있지 않으면 경고 해제

    do {
        let database = try Database(name: "survey")
        let collection = try database.createCollection(name: "survey", scope: "mobile")

        // 기존 문서가 있으면 업데이트, 없으면 새로 저장
        let document = MutableDocument(id: documentID)
        document.setString(gender, forKey: "gender")
        document.setString(age, forKey: "age")
        document.setString(visit, forKey: "visit")
        document.setString(visitReason == "기타" ? otherReason : visitReason, forKey:
            "visitReason")
        document.setString(rating, forKey: "rating")
        document.setString(text, forKey: "text")
        document.setDate(Date(), forKey: "createdAt")

        // 데이터 저장
        try collection.save(document: document)
        print("데이터가 성공적으로 저장되었습니다.")

        // 폼 초기화
        resetForm()
        documentID = ""
    } catch {
        print("데이터 저장 실패: \(error.localizedDescription)")
    }
}
```

1) ID 확인
- 입력한 ID(문서의 이름 같은 것)가 **빈칸**인지 확인
  > ID가 없으면 데이터를 저장할 수 없기 때문에 경고. 메시지
  > 예: "ID를 입력해주세요!"

2) 데이터베이스 준비
- **Mobile Database(name: ”survey")**
  > 이 부분은 **"survey"라는 데이터베이스**를 오픈
- **Collection(name: “survey", scope: ”mobile")**
  > 데이터베이스 안에서 "sync"라는 그룹(Collection)을 생성

3) 문서 작성
- **MutableDocument(id: documentID)**
  ID를 사용해 **문서(Document)**를 생성
  > 문서 = 데이터를 저장할 파일 같은 것
- 문서 안에 정보를 저장
  > gender, age, visit, visitReason, rating, text, 그리고 작성 날짜(createdAt)

4) 데이터 저장
- **collection.save(document: document)**
  문서를 데이터베이스에 저장
  > 새 문서면 추가하고,
  > 기존 문서면 수정
  > 결과: "데이터가 성공적으로 저장되었습니다!"

5) 폼 초기화
- 저장이 끝나면 입력했던 폼을 **초기화**해서 다음 입력을 준비
  > 모든 입력란을 빈칸으로 초기화

3-3. App Connect Code 설명

- SelectDataView.swift

```swift
// 데이터 조회 함수
func fetchData() {
    do {
        let database = try Database(name: "survey")
        let collection = try database.createCollection(name: "survey", scope: "mobile")

        var query: Query

        // 필터 옵션에 따른 쿼리 설정
        if selectedOption == "All" {
            query = QueryBuilder
                .select(SelectResult.all(), SelectResult.expression(Meta.id)) // Meta.id 포함
                .from(DataSource.collection(collection))
        } else {
            query = QueryBuilder
                .select(SelectResult.all(), SelectResult.expression(Meta.id)) // Meta.id 포함
                .from(DataSource.collection(collection))
                .where(Meta.id.equalTo(Expression.string(documentID)))
        }

        // 쿼리 실행 및 결과를 배열로 변환
        let result = try query.execute()
        documents = result.allResults().compactMap { result in
            let dict = result.dictionary(forKey: "survey")
            return MyData(
                id: result.string(forKey: "id") ?? "N/A", // Meta.id 값을 id 필드로 설정
                gender: dict?.string(forKey: "gender") ?? "N/A",
                age: dict?.string(forKey: "age") ?? "N/A",
                visit: dict?.string(forKey: "visit") ?? "N/A",
                visitReason: dict?.string(forKey: "visitReason") ?? "N/A",
                rating: dict?.string(forKey: "rating") ?? "N/A",
                text: dict?.string(forKey: "text") ?? "N/A"
            )
        }

        print("데이터가 성공적으로 조회되었습니다.")
    } catch {
        print("데이터 조회 실패: \(error.localizedDescription)")
    }
```

1) 데이터베이스 준비
- **Mobile Database(name: ”survey")**
  > " survey"라는 데이터베이스를 오픈
- **Collection(name: ”survey", scope: ”mobile")**
  > 데이터베이스 안에서 **survey"라는 그룹(Collection)**을 준비
  > 데이터를 저장한 그룹에서 데이터를 가져옴

2) 쿼리(Query) 준비
- 데이터를 어떻게 가져올지 쿼리(Query)를 생성
- 사용자가 선택한 옵션(selectedOption)에 따라 쿼리가 달라집니다:
  > 모든 데이터를 가져올 때 : selectedOption이 "All"이면 저장된 모든 데이터 가져옴
  > 특정 ID로 데이터를 가져올 때 : selectedOption이 "ID"이면 사용자가 입력한 documentID에 해당하는 데이터만 가져옴
  > 쿼리는 QueryBuilder를 사용해 작성

3) 쿼리 실행
- **query.execute()**
  > 준비한 쿼리를 실행해서 데이터를 가져옴
- 가져온 데이터는 배열로 변환하여(allResults()) 사용하기 쉽게 정리

4) 결과를 객체로 변환
- 가져온 데이터는 MyData라는 객체로 변환
  > 각 데이터의 **필드 값**을 읽어서 객체로 생성
  > 필드 값 예시: id: 문서의 ID, gender: 성별, age: 나이, visit: 방문 여부, visitReason: 방문 이유, rating: 평가 점수, text: 추가 정보

5) 데이터 조회 결과 확인
- 데이터가 성공적으로 조회되면 콘솔에 **"데이터가 성공적으로. 조회되었습니다."** 메시지가 출력됨
- 데이터 조회에 실패하면 오류 메시지를 출력됨

3-3. App Connect Code 설명

- ServerSyncView.swift

```swift
// 데이터 동기화 함수
func syncData() {
    do {
        let database = try Database(name: "survey")
        let collection = try database.createCollection(name: "survey", scope: "mobile")

        // 서버 엔드포인트 설정 (요청하신 URL로 수정)
        let target = URLEndpoint(url: URL(string:
            "wss://u49geqy3uvg9kf6.apps.cloud.couchbase.com:4984/survey")!) //Sync_gateway가
            설치된 서버 주소
        var replConfig = ReplicatorConfiguration(target: target)

        // 컬렉션 및 기본 충돌 해결 방식 추가
        replConfig.addCollection(collection)
        //Couchbase Server에서 생성한 User 정보 RestAPI "Authorization: Basic $DIGEST" 방식과 동일
        replConfig.authenticator = BasicAuthenticator(username: "investigator", password:
            "Passw0rd!")
        replConfig.replicatorType = .pushAndPull

        // 동기화 시작
        let replicator = Replicator(config: replConfig)
        replicator.start()

        replicator.addChangeListener { (change) in
            if let error = change.status.error as NSError? {
                print("동기화 오류 발생: \(error.localizedDescription)")
            } else {
                print("동기화 상태: \(change.status.activity) - 완료 문서 수:
                    \(change.status.progress.completed) / 총 문서 수:
                    \(change.status.progress.total)")
            }
        }

        // 동기화 완료 후 토스트 메시지 표시
        showToast = true
    } catch {
        print("동기화 실패: \(error.localizedDescription)")
    }
```

1) 데이터베이스와 컬렉션 준비
- **Mobile Database(name: "survey")**
  > 로컬에서 사용할 데이터베이스를 오픈
- **Collection(name: "survey", scope: "mobile")**
  > 데이터베이스 안의 특정 그룹(Collection)을 선택
  > 동기화할 데이터가 이 컬렉션에 저장

2) 동기화 서버 설정
- **Sync Gateway 서버 주소**를 설정
- **URLEndpoint(url:"**wss://u49geqy3uvg9kf6.apps.cloud.couchbase.com:4984/survey**")**
  > "wss://"는 동기화를 위한 **WebSocket 프로토콜**
  > u49geqy3uvg9kf6.apps.cloud.couchbase.com 는 App Service 의 DNS 주소.
  > "survey"는 동기화할 Endpoint(Bucket) 이름

3) 동기화 구성
- **ReplicatorConfiguration**으로 동기화 설정:
  > 컬렉션 추가: 동기화할 데이터를 정의
  > 인증 추가: 서버에 접근하기 위해 사용자 이름과 비밀번호 입력.
  > BasicAuthenticator를 사용해 username과 password를 설정
  > 동기화 타입: .pushAndPull을 사용해 **서버로 데이터 보내기**와 **서버에서 데이터 가져오기**를 동시에 설정.

4) 동기화 상태 확인
- **addChangeListener**로 동기화 상태를 실시간으로 확인:
  > 동기화 중 오류가 발생하면 오류 메시지를 출력됨
  > 동기화 진행 상황(완료된 문서 수와 총 문서 수)을 출력합니다.
  예: "완료 문서 수: 5 / 총 문서 수: 10"

5) 동기화 완료 후 알림
동기화가 완료되면 화면에 알림 메시지(토스트)를 표시함

3-4. App 소스 Endpoint 설정

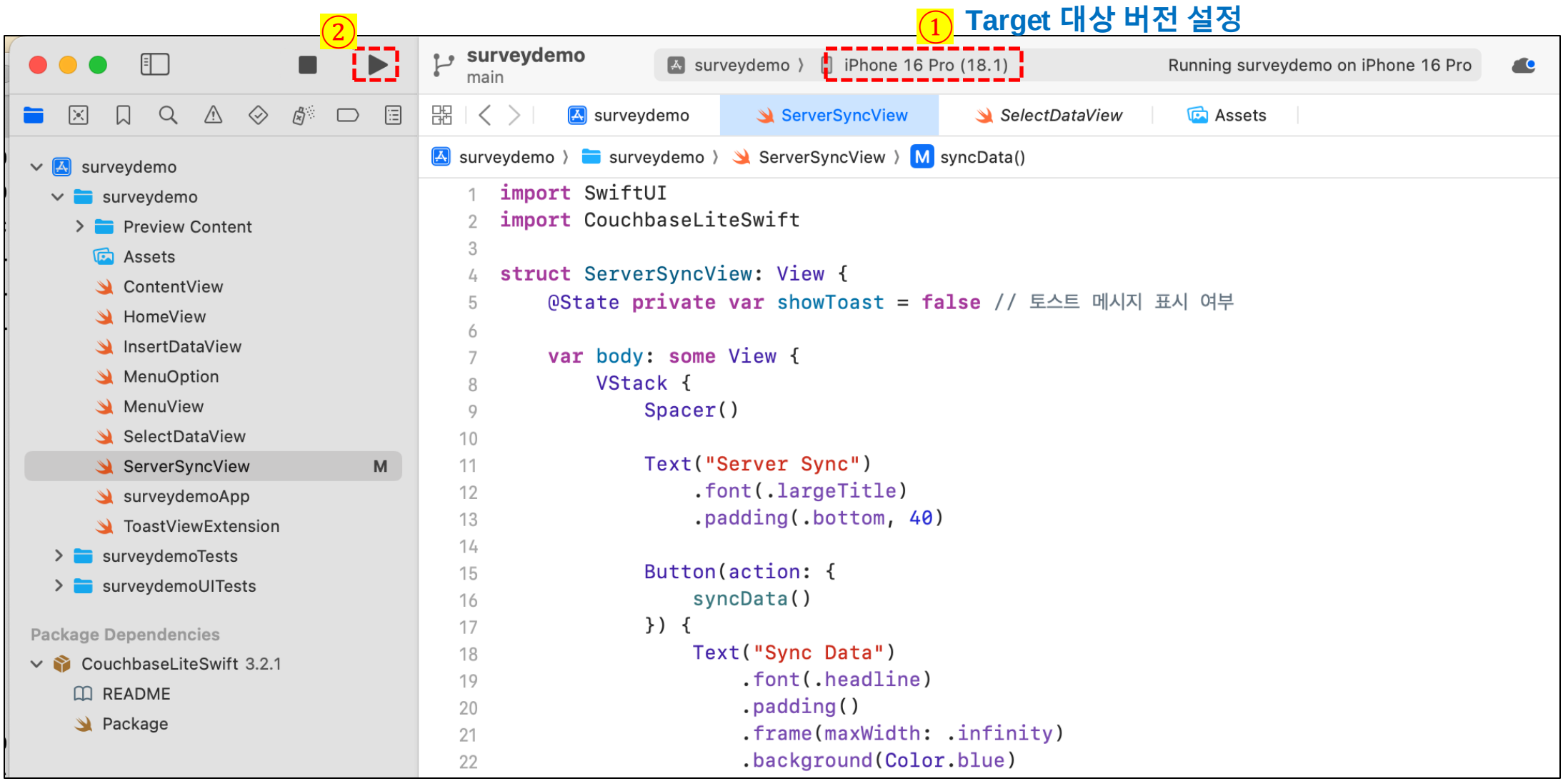
- ServerSyncView.swift

```swift
// 데이터 동기화 함수
func syncData() {
    do {
        let database = try Database(name: "survey")
        let collection = try database.createCollection(name: "survey", scope: "mobile")

        // 서버 엔드포인트 설정 (요청하신 URL로 수정)
        let target = URLEndpoint(url: URL(string:
            "wss://u49geqy3uvg9kf6.apps.cloud.couchbase.com:4984/survey")!) //Sync_gateway가
            설치된 서버 주소
        var replConfig = ReplicatorConfiguration(target: target)

        // 컬렉션 및 기본 충돌 해결 방식 추가
        replConfig.addCollection(collection)
        //Couchbase Server에서 생성한 User 정보 RestAPI "Authorization: Basic BDIGEST" 방식과 동일
        replConfig.authenticator = BasicAuthenticator(username: "investigator", password:
            "Passw0rd!")
        replConfig.replicatorType = .pushAndPull

        // 동기화 시작
        let replicator = Replicator(config: replConfig)
        replicator.start()

        replicator.addChangeListener { (change) in
            if let error = change.status.error as NSError? {
                print("동기화 오류 발생: \(error.localizedDescription)")
            } else {
                print("동기화 상태: \(change.status.activity) - 완료 문서 수:
                    \(change.status.progress.completed) / 총 문서 수:
                    \(change.status.progress.total)")
            }
        }

        // 동기화 완료 후 토스트 메시지 표시
        showToast = true
    } catch {
        print("동기화 실패: \(error.localizedDescription)")
    }
```

1) Sync_gateway를 통한 App 연결
> Sync Gateway App IP 주소 + Endpoint(Bucket) 명
> Sync_gateway와 Capella AppService는 비밀번호 요구사항이 다르기 때문에 Password 확인

2) AppService를 통한 App 연결
> AppService Connect Tab에서 사용 되는 IP 사용
> 허용 IP 설정
> Sync_gateway와 Capella AppService는 비밀번호 요구사항이 다르기 때문에 Password 확인

3-5. App 빌드 및 실행



Hands-on Lab

> On-Mobile AI

Semantic Search / GenAI Demo

1. **Couchbase Lite Vector Search Sample Apps (Color / Word)**
   https://github.com/couchbaselabs/couchbase-lite-vector-search-samples

2. **Simple Intelligence - vector search with Apple Vision AI**
   https://github.com/waynecarter/simple-intelligence

3. **WhatIsThisThing - vector search with Apple Vision AI**
   https://github.com/mgroves/WhatIsThisThing

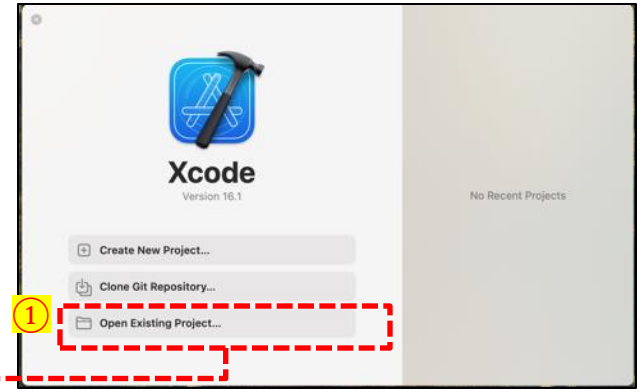
참고 : Couchbase Mobile 샘플
https://github.com/waynecarter/simple-sync
https://github.com/unixfree/surveydemo
https://github.com/unixfree/userprofiledemo
https://docs.couchbase.com/home/index.html

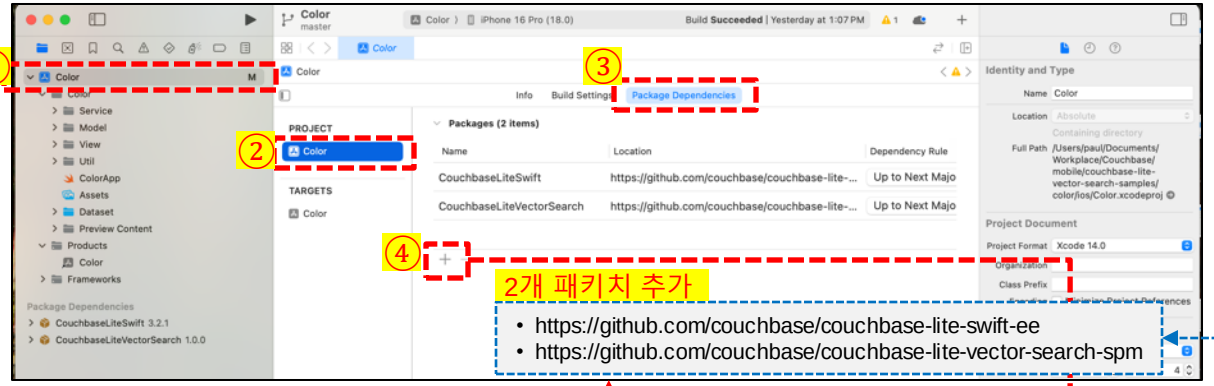
Vector Search Sample Apps (Color / Word)

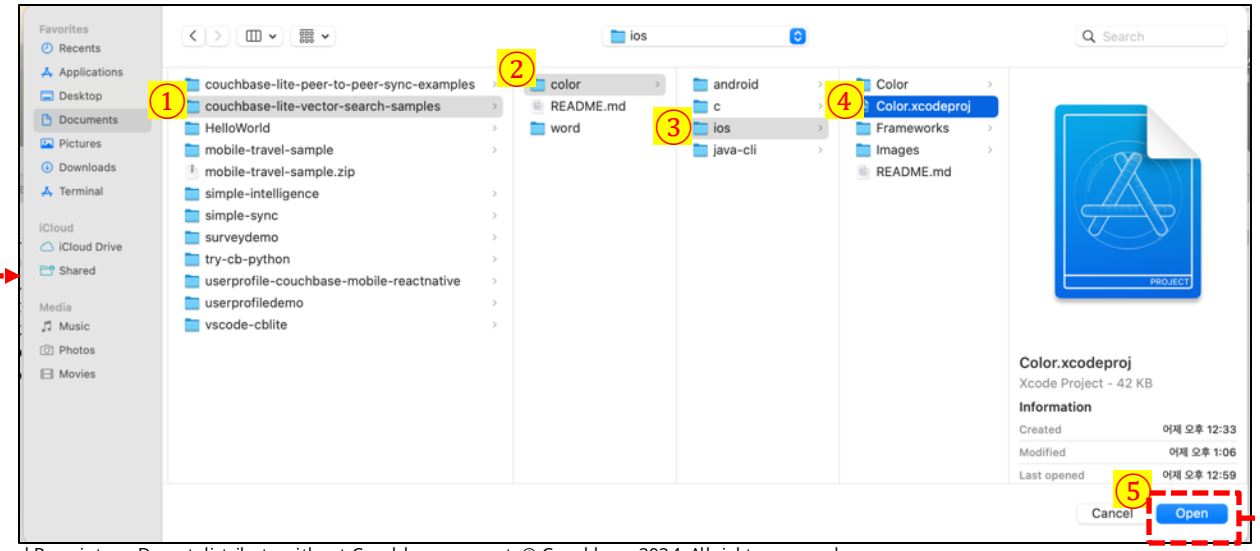
https://github.com/couchbaselabs/couchbase-lite-vector-search-samples

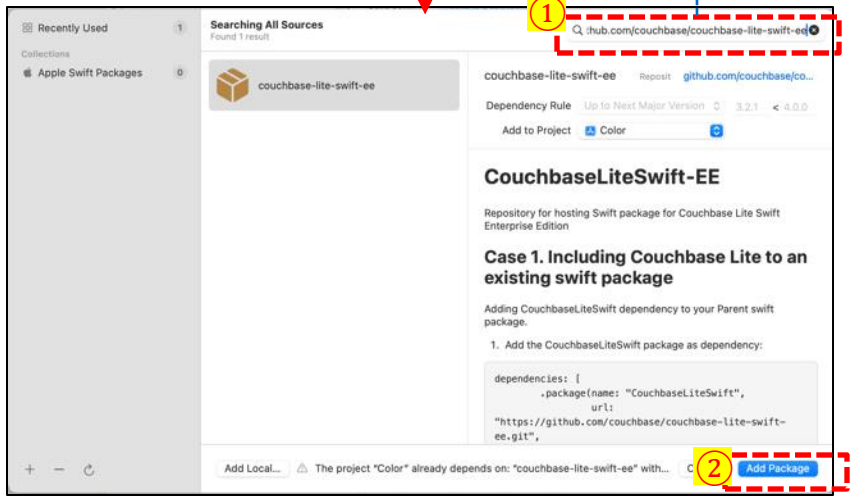
```
$ git clone https://github.com/couchbaselabs/couchbase-lite-vector-search-samples.git
$ cd couchbase-lite-vector-search-samples
```

Xcode 실행 후, Open Existing Project **선택 후 > .. .. > Color.xcodeproj > Open 클릭**









Vector Search Sample Apps (Color / Word)

Word > AppService

```
/// Initialize the database by loading the dataset into the database and creating a vector index.
func initialize() throws {
    if (initialized) { return }

    // Load dataset:
    db = try self.loadDataset()

    // Get the words collection:
    collection = try db.collection(name: kCollectionName)!

    // Create a vector index:
    var config = VectorIndexConfiguration(expression: "vector", dimensions: 300, centroids: 8)
    config.metric = .cosine
    try collection.createIndex(withName: kIndexName, config: config)

    initialized = true
}

/// Search similar words.
func search(_ word: String) throws -> [Word] {
    if (!initialized) { throw "Service is not initialized" }

    // Use Apple's Word Embedding ML model to get a vector for the word:
    guard let wordVector = model.vector(for: word.lowercased()) else {
        throw "No vector for '\(word)'"
    }

    // Create a query object:
    if (query == nil) {
        let sql = "SELECT meta().id, word, VECTOR_DISTANCE(\(kIndexName)) " +
                  "FROM \(collection.fullName) " +
                  "WHERE VECTOR_MATCH(\(kIndexName), $vector, 10)"
        query = try db.createQuery(sql)
    }
```

Color > AppService

```
/// Initialize the database by loading the dataset into the database and creating a vector index.
func initialize() throws {
    if (initialized) { return }

    try Extension.enableVectorSearch()

    // Load dataset:
    db = try self.loadDataset()

    // Get the colors collection:
    collection = try db.collection(name: kCollectionName)!

    // Create a vector index:
    var config = VectorIndexConfiguration(expression: "colorvect_l2", dimensions: 3, centroids: 2)
    config.encoding = VectorEncoding.none
    try collection.createIndex(withName: kIndexName, config: config)

    initialized = true
}

/// Search similar colors.
func search(_ color: [Int]) throws -> [ColorObject] {
    if (!initialized) { throw "Service is not initialized" }

    // Create a query object:
    if (query == nil) {
        let sql = "SELECT id, color, colorvect_l2, APPROX_VECTOR_DISTANCE(colorvect_l2, $vector) " +
                  "FROM \(collection.fullName) " +
                  "LIMIT 8"
        query = try db.createQuery(sql)
    }
```

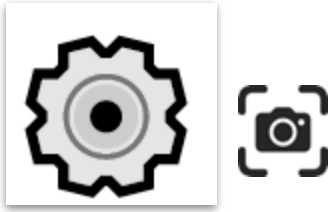
https://www.youtube.com/watch?v=8_O0ZTrmi30

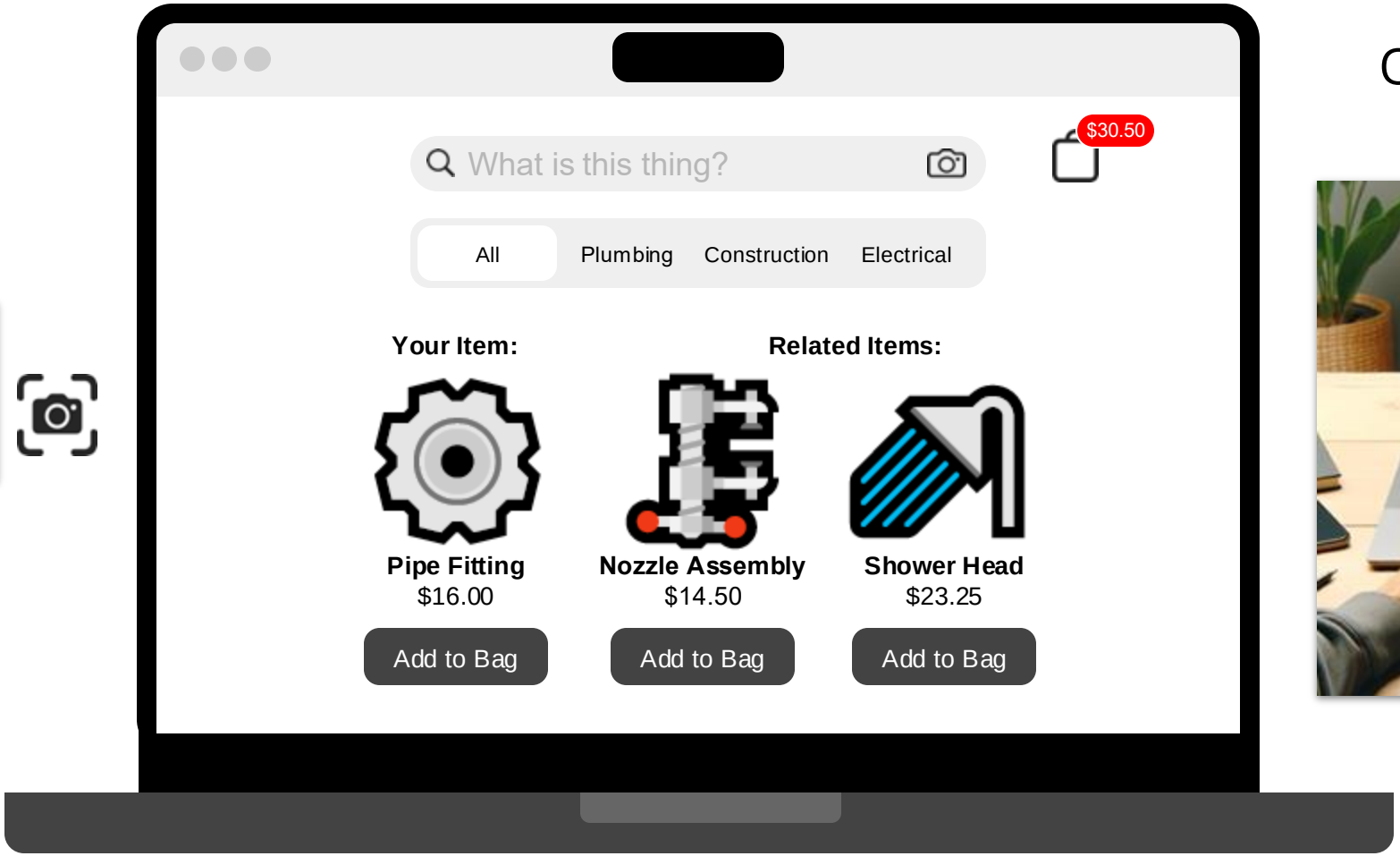
Simple Intelligence - vector search with Apple Vision AI

- **Simple Intelligence App**

https://apps.apple.com/us/app/simple-intelligence/id6504311724

Capella Vector Search For Web Apps | What Is This Thing?





Online Shopping (itemfinder)

[Demo video](#)

[GitHub repo](#)

And then …

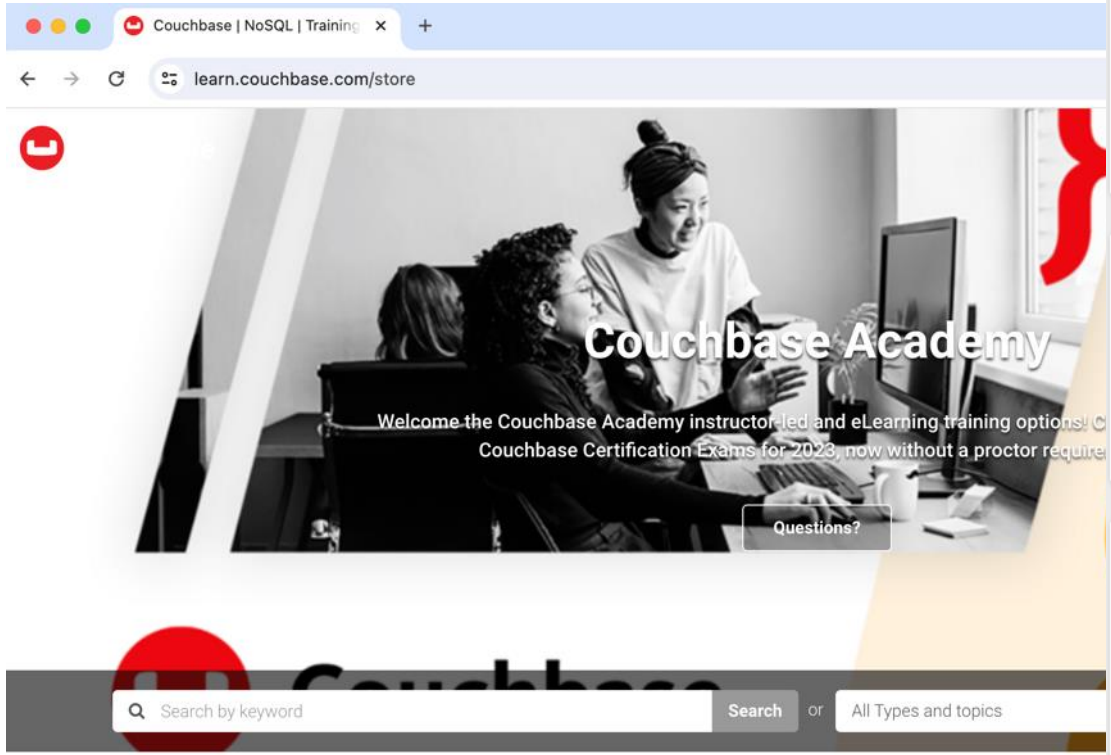
Next Capella Webinar

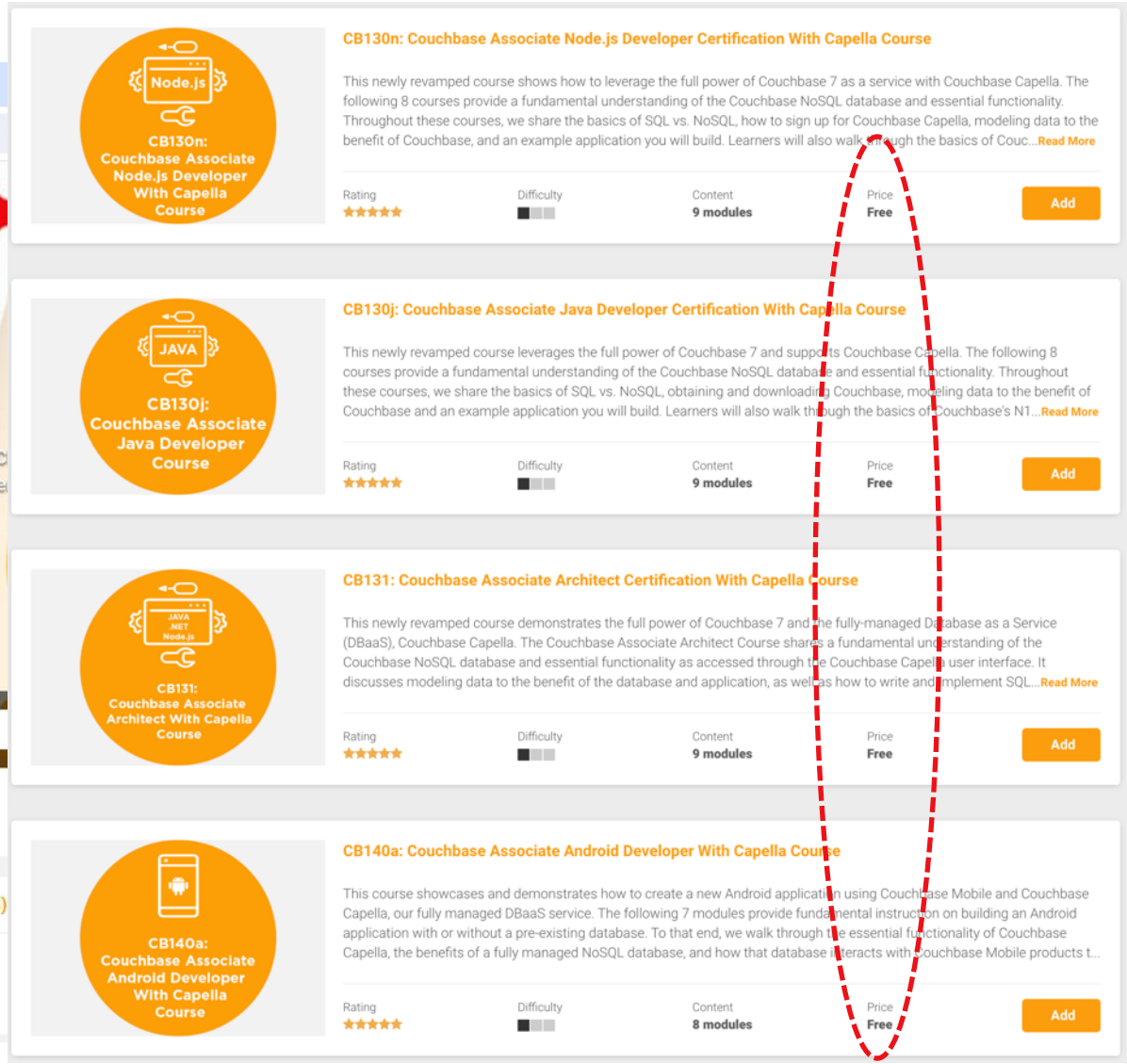
구분	일자	Webinar 주제
Feature 시리즈1	2025-02-25	Capella Columnar Service 소개
SDK 시리즈1	2025-03-26	Couchbase 기반 3 Tier Application 개발 실습, Java
AI 시리즈1	2025-04-30	Couchbase 기반 AutoRAG 구축, Python

Couchbase Academy

https://docs.couchbase.com/
https://developer.couchbase.com/
https://couchbase.live/
https://query-tutorial.couchbase.com/tutorial/#1

https://learn.couchbase.com/store





Thank you!

Paul.Son@couchbase.com

www.couchbase.com

cloud.couchbase.com

Couchbase

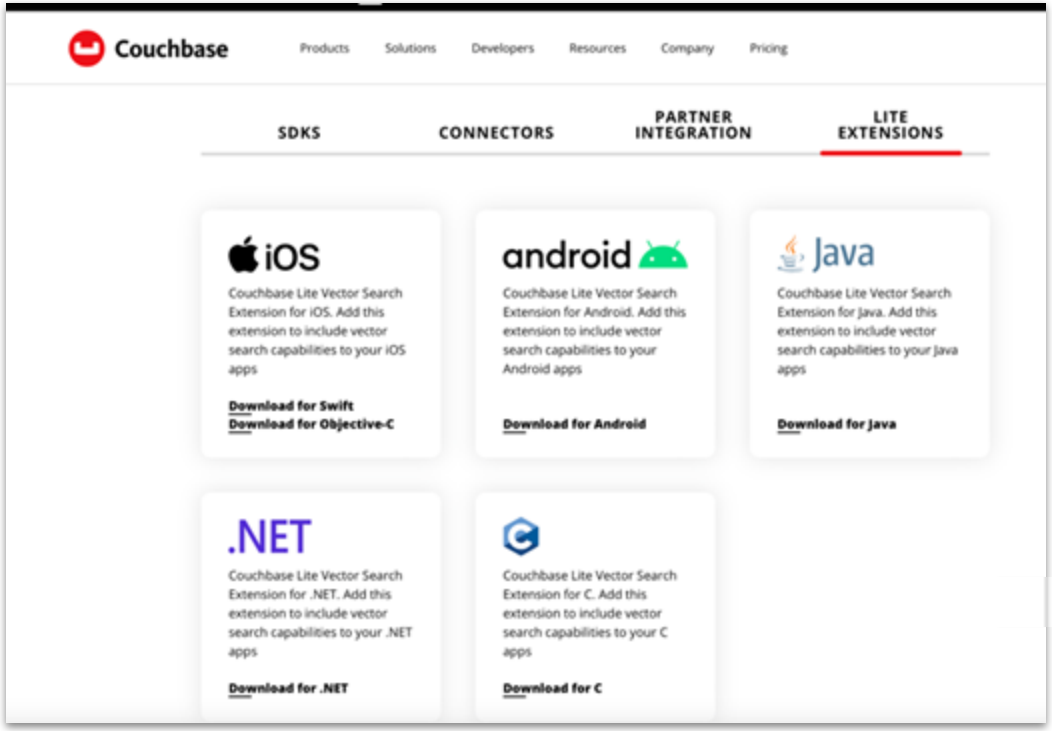
Vector Search with Couchbase Lite

Packaging and Code Samples

Vector Search Packaging

Vector Search is EE only offering

- Distributed as a standalone extensions library separate from the main Couchbase Lite SDK
- The library size depends on platform - can be 10s of MB
- Applications that do not need this feature do not have to include it
- Application must link both Couchbase Lite and Extensions library if they want to use vector search functionality



Vector Index Configuration

Setting	Description	Default value
expression	Any SQL++ expression identifying what needs to be indexed. In simplest case, this would be a document property for instance.	
dimensions	• 2 - 4096	
metric	• euclideanSquared • cosine (1 - cosine similarity) • euclidean • dot (negated dot product)	Default : euclideanSquared
number of centroids	• 1 - 64000	
encoding	• None: No compression, no data loss • Scalar Quantizer (SQ) | SQ-8 bits: Reduce # of bits per dimension • Product Quantizer (PQ): Reduce # of dimensions and bits per dimension	Default: Scalar Quantizer (SQ) | SQ-8 bits
minTrainingSize **maxTrainingSize**	• Min: Determined by index based on #centroids & encoding parameters • Max: Determined by index based on #centroids & encoding parameters	Default : 0
numProbes	• # of centroid buckets that the query will search for the nearby vectors	Default : 0 Auto calculated based on #centroids
isLazy	• Enable the vector index in lazy mode	Default : false

Creating Vector Index

Creating a vector index in Couchbase Lite is very simple, yet powerful offering a number of configurable options to meet your application needs

```
1  var config = VectorIndexConfiguration(expression: "description", dimensions: 1028, centroids: 20)

2  config.metric = .cosine
   config.minTrainingSize = 50
   config.maxTrainingSize = 200

3  try collection.createIndex(withName: "myIndex", config: config)
```

1. Create a vector index configuration with the expression identifying what needs to be indexed, the dimensions of the vector, and number of centroids.
   - In this example, a vector index on the “description” property of the documents is configured.
2. Set the optional parameters of the vector index configuration, such as metric, minTrainingSize and maxTrainingSize
3. Create the vector index with the configuration in the target collection

Vector Search Query with SQL++

APPROX_VECTOR_DISTANCE(vector-expr, target-vector [, metric] [, nprobes] [, accurate])

Parameter	Description	Comment
vector-expr	• The expression returning a vector • Must match the expression specified in the vector index	The expression is NOT the index name.
target-vector	• The target vector	
metric	Values: • `"EUCLIDEAN_SQUARED"` • `"L2_SQUARED"` • `"EUCLIDEAN"` • `"L2"` • `"COSINE"` • `"DOT"`	OPTIONAL • If not specified, the metric set in the vector index is used. • If specified, the metric must match with the metric set in the vector index.
nprobes	• Number of buckets to search for the nearby vectors	OPTIONAL • If not specified, the nprobes set in the vector index is used.
accurate	If not present, false will be used, which means that the quantized / encoded vectors in the index will be used for calculating the distance.	OPTIONAL • Only accurate = false is supported

Vector Search Query with SQL++

Run a vector search query against the index in just a few easy steps

```
1  guard let searchEmbedding = modelRef.getEmbedding(for: searchPhrase) else { throws Errors.notFound }
2  let sql = "SELECT meta().id, description FROM _ ORDER BY APPROX_VECTOR_DISTANCE(description, $searchParam) LIMIT 10"
3  let query = try db.createQuery(sql)
4  let params = Parameters()
   params.setValue(searchEmbedding, forName: "searchParam")
   query.parameters = params
5  try query.execute()
```

1. Get embedding vector for search Phrase from embedding model
2. Construct the SQL++ query to return documents with content similar to searchParam
3. Construct the vector search query
4. Set the embedding vector associated with the search parameters in the vector search query
5. Execute the vector search query

Hybrid Search Query with SQL++ | Example 1

Construct powerful SQL++ queries by combining vector search with regular search or other regular filtered queries

```
SELECT docs
FROM database
WHERE MATCH(ftsIndex, $text) AND (group = 'group1' OR group = 'group2')
ORDER BY APPROX_VECTOR_DISTANCE(vector, $searchParam)
```

1. Documents matching the filter criteria as specified in WHERE clause is determined.
    - In this example, FTS search is performed on documents to find matching items according to MATCH criteria.
    - Documents are then filtered based on the filter criteria specified in WHERE clause is applied.
2. Semantic search is then performed on the filtered set of documents to find items similar to the search param.
3. The list of documents is returned.

Hybrid Search Query with SQL++ | Example 2

Construct powerful SQL++ queries by combining vector search with regular search or other regular filtered queries

```
3  SELECT docs
   FROM database
1  WHERE (group = 'group1' OR group = 'group2')
2        AND APPROX_VECTOR_DISTANCE(vector, $searchParam) < $threshold
   LIMIT 10
```

1. Documents are filtered based on the filter criteria specified in WHERE clause.
2. Semantic search is then performed on the filtered set of documents to find items similar to the search param, retrieving only documents that are within the expected distance.
3. The top N matching documents as specified in LIMIT clause are returned.

Using Prediction function for generating embeddings (1/2)

When documents do not have to embeddings inline, embeddings are created dynamically via `predict()`

```
class myEmbeddingModel: PredictiveModel {
        func predict(input: DictionaryObject) -> DictionaryObject? {
1       guard let searchStr = input.string(forKey: "myTextKey") else { return nil }
2       guard let vector = embeddingModel.getEmbedding(for: searchStr) else { return nil }
        let result = MutableDictionaryObject()
3       result.setValue(vector, forKey: "vector")
        return result
    }

    func registerEmbeddingModel() {
4       Database.prediction.registerModel(myEmbeddingModel(), withName: "myPredictModel")
    }
}
```

1. Get content for which we need to get embedding passed in via dictionary parameter
2. Fetch embedding from model
3. Set embedding vector in return back to caller
4. Register predictive model with Couchbase Lite

Using Prediction function for generating embeddings (2/2)

Vector embedding that is returned from `prediction()` function for specified content is indexed

1
```
var config = VectorIndexConfiguration(expression: "prediction('myPredictModel', {\"myTextKey\":description}).vector",
                                                                    dimensions: 300, controids: 20)

config.metric = .cosine
config.minTrainingSize = 50
config.maxTrainingSize = 200

try collection.createIndex(withName: "myIndex", config: config)
```

1. The expression calls into the `prediction` function which returns vector embedding for the specified document property. The `"description"` is the property in document to vectorize, sent in via input dictionary.

2
```
SELECT docs
FROM database
ORDER BY APPROX_VECTOR_DISTANCE(prediction('myPredictModel', {\"myTextKey\": description}).vector, $searchParam)
LIMIT 10
```

2. APPROX_VECTOR_DISTANCE runs vector search query against embedding for searchParam

Using Lazy Vector Index for generating embeddings

Applications are in full control over when vector embeddings are generated

```
1  let index = try collection.getIndex(name: "myvectorIndex")

   while true {
2    guard let updater = try index.beginUpdate(limit: 100)
     else { break // (updater is null) }

     for i in 0..< updater!.count {
        let embedding: [Float]? = nil
3       let content = updater!.string(at: i)
        do {
4           embedding = try await model.getEmbedding(text: content)
        } catch modelError.NetworkError {
5           try updater!.skipVector(at: i)
                        continue
        } catch {
                  vector = nil
        }
6       try updater!.setVector(embedding, at: i)
     }
7    try updater.finish()
   }
```

1. Get handle to vector Index
2. Begin updating index in chunks of up to 100 vectors by invoking the updater
3. Get the document property to vectorize
4. Invoke model to fetch embedding for specified content
5. If error in fetching embedding from model, skip that
6. Return the embedding to index updater
7. Commit the vector index updates