데이터사이언스를위한지식관리시스템

10주차: MongoDB

MongoDB

- 간단히 몇 개의 단어로만 표현하면 JSON document를 저장하는 데이터베이스라고 할 수 있습니다
- JSON document를 사용한다는 것은 schema에 대해 유연함을 의미합니다
- 이러한 의미로 Not Only SQL (NoSQL) 이라고도 합니다

- JSON 데이터를 우리가 관리하는 파일로 기록하지 않고 이러한 JSON document를 관리하는 MongoDB에 기록해보면 어떨까요?
- JSON 의 관리를 MongoDB라는 곳에 맡기는 것입니다

MongoDB의 collection과 document

- collection: RDBMS의 테이블과 비슷한 역할을 합니다
- document: RDBMS의 레코드와 비슷한 역할을 합니다
- 따라서 RDBMS의 레코드를 MongoDB에 옮겨 넣는 것은 가능합니다

- RDBMS로 생각하면 매 수집되는 시계열 데이터를 각각의 레코드로 하여 테이블에 적재하게 되겠습니다
- 그러나 우리는 MongoDB에서 조금 다른 접근을 해보고자 합니다
- JSON 이라는 자료형이 가지는 장점을 활용해보는 것입니다

- https://www.mongodb.com/ko-kr/docs/manual/core/databases-and-collections/ 를 참고합니다(MongoDB는 데이터 기록을 문서(특히 BSON 문서)로 저장하며, 이 문서들은 컬렉션에 함께 모여 있습니다. 데이터베이스는 하나 이상의 문서 컬렉션을 저장합니다.)
- BSON = binary json 으로 *.json 파일 자체는 텍스트파일로 기록되는 것에 반해 BSON은 binary 형태로 기록됩니다

On-Premise형 설치와 Managed Service

- 직접 설치해서 쓸지, 누군가 설치해준 인스턴스만 사용할지의 차이 입니다
- 전자는 서버에 직접 접근 가능하고 후자는 서비스 인스턴스에만 접근 가능합니다

- On-Premise는 우리가 ssh에 직접 접근해서 설치명령을 내리는 형태로 생각하시면 됩니다
- Kafka를 예로 들면 우리가 진행한 것이 On-Premise 설치입니다
- Kafka도 Managed Service가 있는데 https://cloud.google.com/products/managed-service-for-apache-kafka?hl=ko 또는 https://aws.amazon.com/ko/msk/ 와 같은 것을 생각하시면 됩니다

- AWS Managed Kafka의 예를 들면 https://docs.aws.amazon.com/msk/latest/developerguide/create-topic.html 와 같이 Topic을 생성하고 https://docs.aws.amazon.com/msk/latest/developerguide/produce-consume.html 와 같이 Produce/Consume 하는 것이 우리가 했던 것과 정확히 같습니다
- 즉, On-Premise형인지 Managed Service인지는 Client 측면에서는 차이가 없다고 보아도 무방합니다

MongoDB의 On-Premise와 Managed Service

- MongoDB의 경우 Public Cloud에서 Managed Service를 직접 제공하지 않고 MongoDB(Atlas)측에서 직접 Public Cloud에 provisioning하여 서비스 됩니다
- 따라서 MongoDB Managed Service 사용시 provisioning될 Public cloud를 선택할 수 있습니다

- On-Premise 형태, 또는 클라우드에 직접 구축 형태의 설치는 https://www.mongodb.com/ko-kr/docs/manual/administration/install-community/ 를 따르면 됩니다
- Managed Service 형태의 provisioning은 https://www.mongodb.com/ko-kr 에서 계정을 만들고 클러스터 생성을 진행하면 됩니다

- 우리는 On-Premise 형태, 또는 클라우드에 직접 구축 형태로 진행합니다

MongoDB 의 provisioning - Atlas on public cloud

- Managed Service이기 때문에 서버는 우리가 별도로 우리의 PC에 설치하지 않습니다
- 우리는 별도로 MongoDB를 설치할 필요 없이 Atlas에서 MongoDB cluster 생성을 요청하고 접속하면 됩니다

- 우리가 준비할 것은 MongoDB Client 입니다
- ***서버는 설치하지 않지만 클라이언트는 설치합니다***

MongoDB Client

- Kafka broker에 메시지를 쓰기위한 client는 Kafka producer 였습니다
- Kafka broker에 있는 메시지를 쓰기위한 client는 Kafka consumer 였습니다

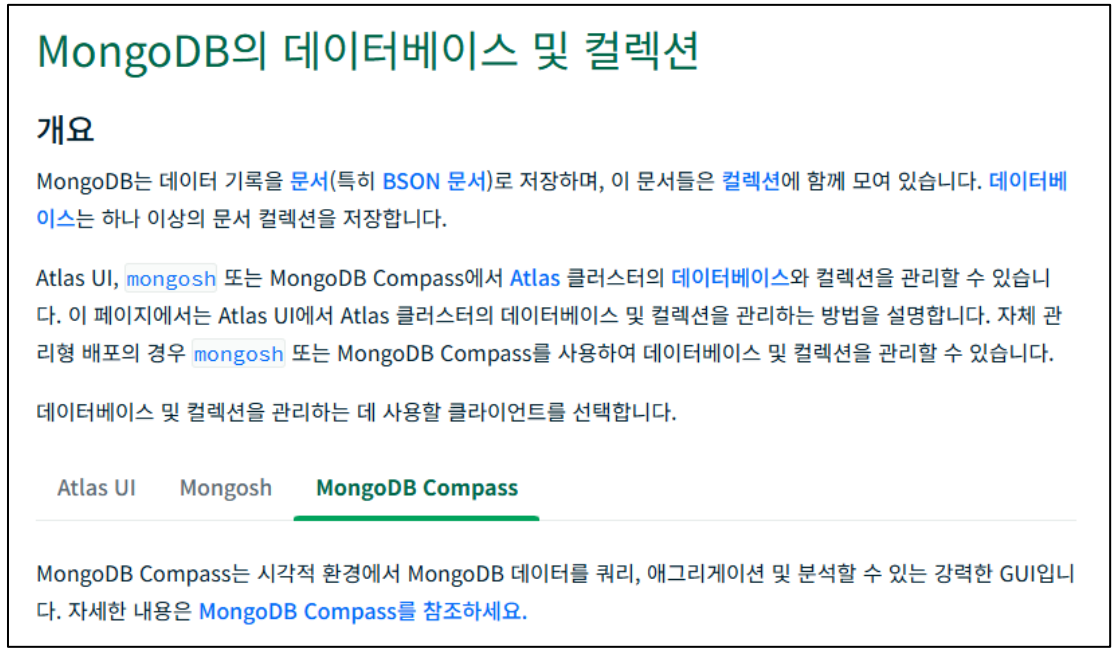
- 우리는 두가지 형태로 Kafka client를 사용해보았습니다
- CLI: kafka-console-producer.sh, kafka-console-consumer.sh
- Python connector
- https://github.com/provectus/kafka-ui 와 같은 GUI 형태의 client도 있습니다

- 우리는 두가지 형태로 MongoDB client를 사용하겠습니다
- GUI: MongoDB compass
- Python connector

MongoDB Compass

- GUI형태의 MongoDB Client

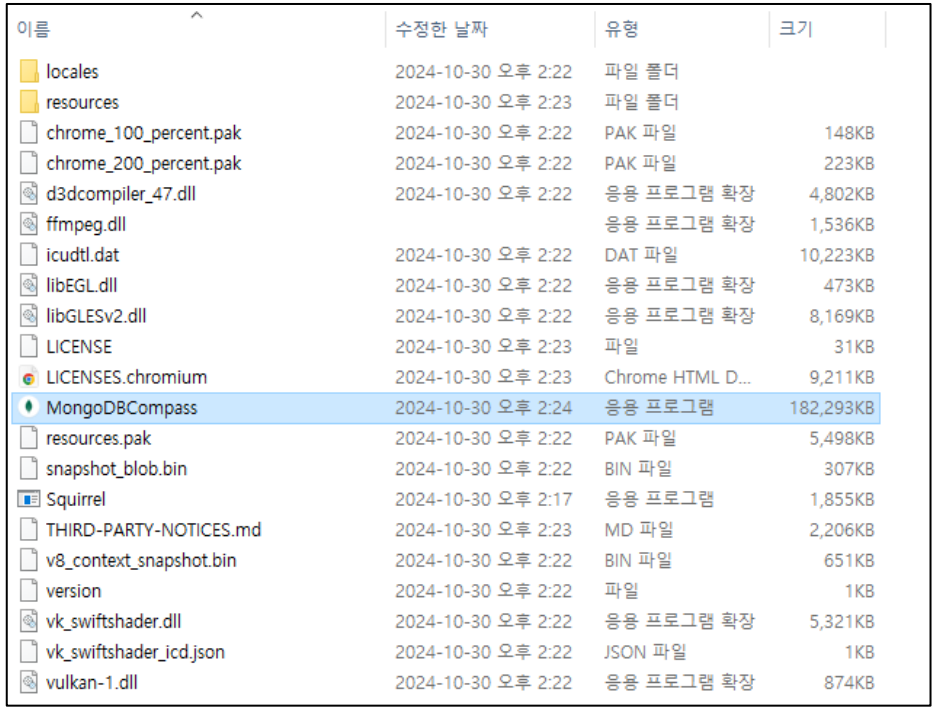
- 다른 Client를 사용해도 프로토콜은 동일합니다
- Client UI가 다를 뿐입니다



- 웹브라우저를 통해서도 RestAPI 수집이 가능하지만 python으로도 수집이 가능한것과 같습니다
- GUI형태의 Client를 통해서도 MongoDB 접근이 가능하지만 python으로도 접근이 가능합니다
- 우리는 python을 이용해 write 작업을 수행하고 GUI를 통해 read 작업을 수행합니다

MongoDB compass 다운로드와 실행

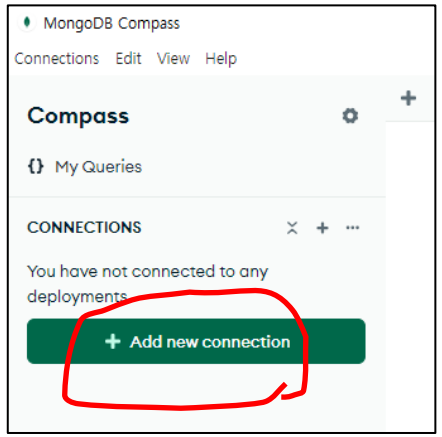
- https://www.mongodb.com/try/download/compass 에서 사용하시는 OS에 맞게 선택합니다
- For MS Windows(Intel): https://downloads.mongodb.com/compass/mongodb-compass-1.44.6-win32-x64.zip
- For macOS(Sillicon): https://downloads.mongodb.com/compass/mongodb-compass-1.44.6-darwin-arm64.dmg
- MS Windows 사용자의 경우 압축을 새 디렉토리에 풀고 MongoDBCompass.exe 파일을 실행합니다

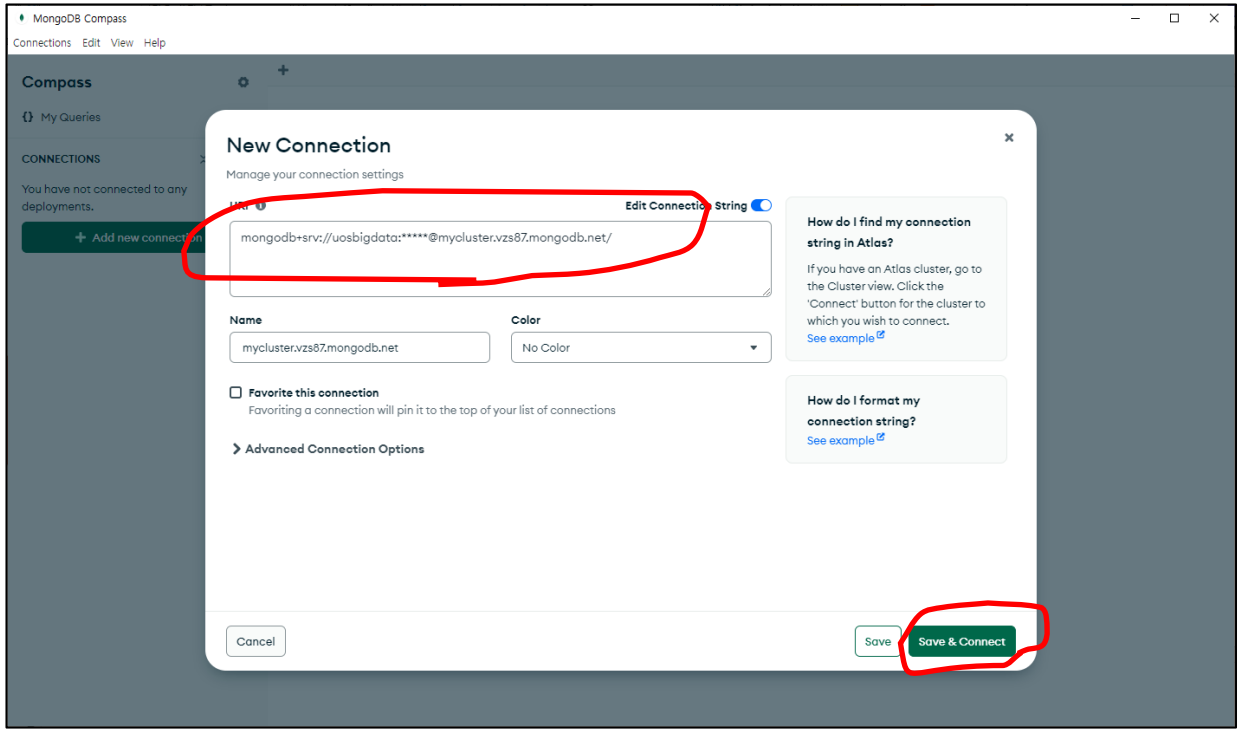


이름	수정한 날짜	유형	크기
locales	2024-10-30 오후 2:22	파일 폴더	
resources	2024-10-30 오후 2:23	파일 폴더	
chrome_100_percent.pak	2024-10-30 오후 2:22	PAK 파일	148KB
chrome_200_percent.pak	2024-10-30 오후 2:22	PAK 파일	223KB
d3dcompiler_47.dll	2024-10-30 오후 2:22	응용 프로그램 확장	4,802KB
ffmpeg.dll		응용 프로그램 확장	1,536KB
icudtl.dat	2024-10-30 오후 2:22	DAT 파일	10,223KB
libEGL.dll	2024-10-30 오후 2:22	응용 프로그램 확장	473KB
libGLESv2.dll	2024-10-30 오후 2:22	응용 프로그램 확장	8,169KB
LICENSE	2024-10-30 오후 2:23	파일	31KB
LICENSES.chromium	2024-10-30 오후 2:23	Chrome HTML D...	9,211KB
MongoDBCompass	2024-10-30 오후 2:24	응용 프로그램	182,293KB
resources.pak	2024-10-30 오후 2:22	PAK 파일	5,498KB
snapshot_blob.bin	2024-10-30 오후 2:22	BIN 파일	307KB
Squirrel	2024-10-30 오후 2:17	응용 프로그램	1,855KB
THIRD-PARTY-NOTICES.md	2024-10-30 오후 2:23	MD 파일	2,206KB
v8_context_snapshot.bin	2024-10-30 오후 2:22	BIN 파일	651KB
version	2024-10-30 오후 2:22	파일	1KB
vk_swiftshader.dll	2024-10-30 오후 2:22	응용 프로그램 확장	5,321KB
vk_swiftshader_icd.json	2024-10-30 오후 2:22	JSON 파일	1KB
vulkan-1.dll	2024-10-30 오후 2:22	응용 프로그램 확장	874KB

MongoDB Compass로 MongoDB Cluster 접속

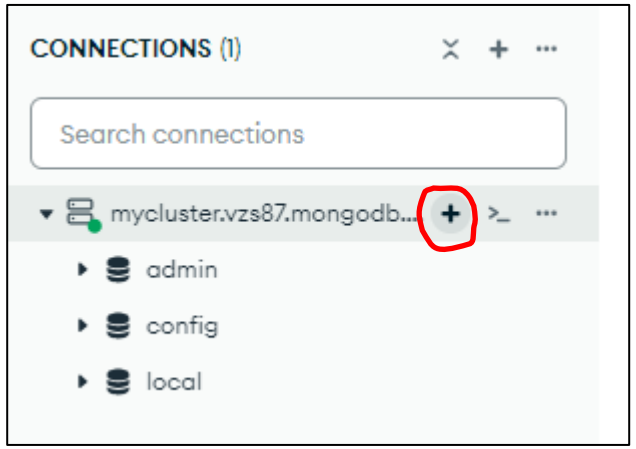
- MongoDB Connection String으로부터 MongoDB Cluster의 IP가 지정된다는 것을 확인했으니 이제 접속을 해봅니다
- Connection String scheme은 아래와 같습니다
- mongodb://localhost:27017
- <db_password> 패스워드를 실제 패스워드로 바꾸고 연결을 시도합니다

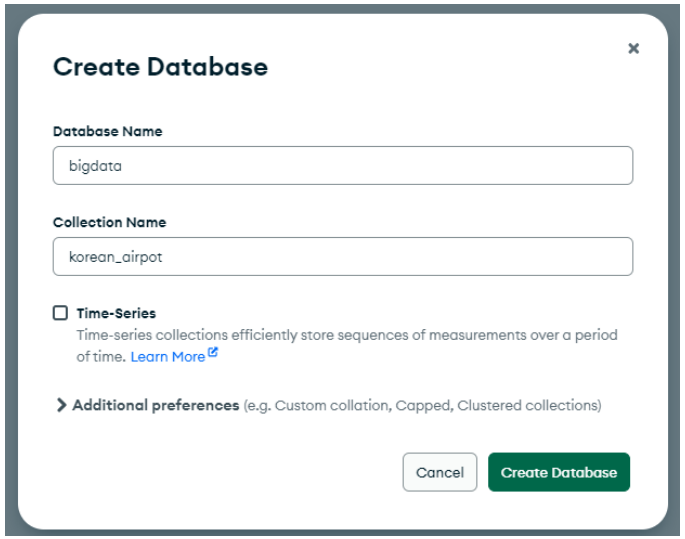


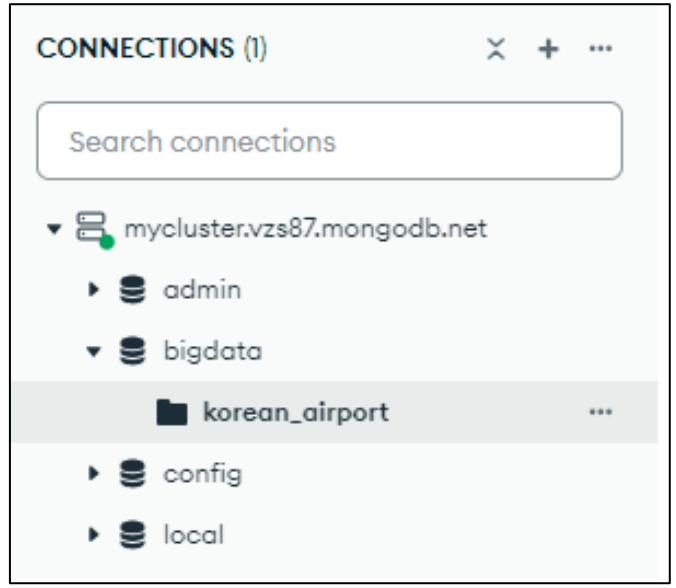


MongoDB Database 만들기

- CONNECTIONS에 1개의 연결과 3개의 Database를 확인할 수 있습니다
- 연결정보에 있는 + 버튼을 누르면 Database를 만들 수 있습니다
- Database를 만들 때에는 collection 까지 함께 생성하게 됩니다

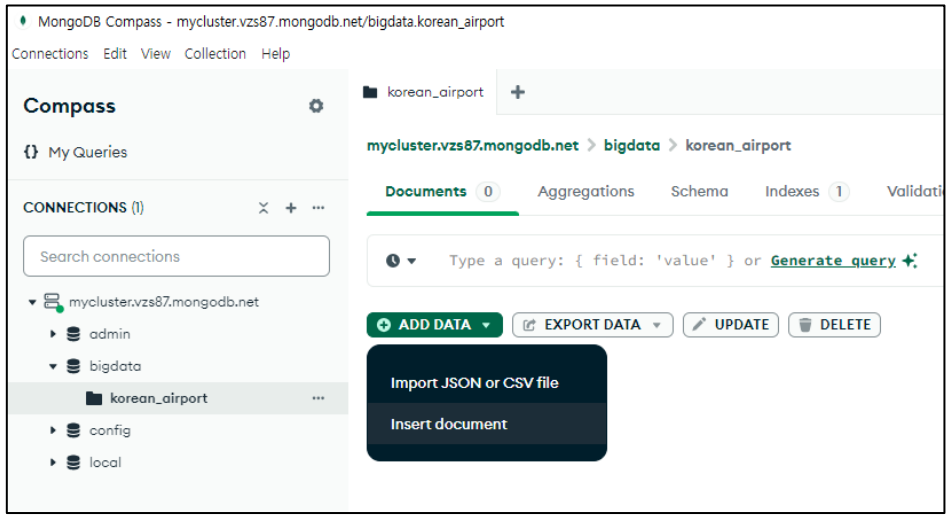


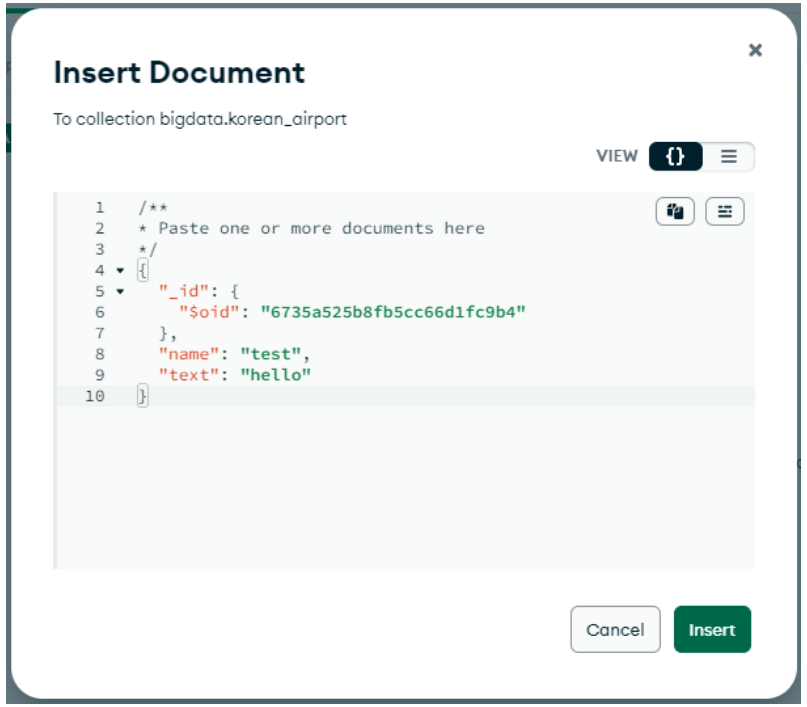




Collection에 Document 추가하기

- ADD DATA > Insert document 를 선택합니다
- MongoDB Compass UI에서 직접 document를 생성하는 방식 입니다
- 아래 oid는 수정하지 않습니다





MongoDB document의 oid

- Document의 고유성을 보장하기 위한 ID 입니다
- 기본 인덱스가 걸려 있습니다
- https://www.mongodb.com/ko-kr/docs/manual/reference/method/ObjectId/ 에서 oid의 구조를 확인할 수 있습니다
- 위 문서의 "12바이트 ObjectId는 다음으로 구성됩니다." 를 참고합니다
- Object ID로 document의 timestamp 를 관리하는 것이 best practice는 아닙니다
- 그러나 정말 필요할 때에는 이 정보를 활용할 수 있습니다

```
{
  "_id": {
    "$oid": "6735a5dbb8fb5cc66d1fc9b5"
  }
}
```

`6735a5db` Timestamp: 1731569115 = 2024년 11월 14일 목요일 오후 4:25:15

`b8fb5cc66d` 프로세스 고유 ID: 동일 collection에 접근하는 다른 프로세스와의 중복 방지

`1fc9b5` 임의의 초기화 값으로부터의 증분: 동일 timestamp, 동일 프로세스 내에서의 중복 방지

MongoDB python connector 사용

- pymongo 패키지를 설치합니다
- MongoDB Compass 대신 사용하는 라이브러리 입니다

```
pip install pymongo
```

- MongoDB Compass와의 차이점은 UI의 차이이며 MongoDB와의 통신 프로토콜은 동일함을 의미합니다
- 따라서 동일한 Connection string으로 연결합니다
- 연결된 후에는 Database > Collection 순서로 접근하는 것 또한 동일합니다

```python
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")

db = client["bigdata"]
collection = db["korean_airport"]
```

Python으로 MongoDB collection에 데이터 쓰기

- Document는 collection 내에 read/write 되는 것도 동일합니다
- 아래 코드를 반복 실행하면 ObjectId는 어떻게 될까요?
- 결과는 ObjectId 내의 프로세스 고유 ID는 유지되는 상태로 Timestamp 필드와(상위 4bytes), 임의 증분 데이터(하위 3bytes)가 변경됩니다

```
doc = {'name': 'test', 'text': 'hello'}

collection.insert_one(doc)
```

X 3

Python으로 MongoDB collection에 데이터 쓰기

- 여러 개의 데이터를 한번의 요청으로 기록하는 방법은 아래와 같습니다
- 이 때 Timestamp는 동일하게 임의 증분 필드가 각각 +1씩 되는 것을 확인할 수 있습니다
- 즉 요청된 시각은 동일하고 처리하는 프로세스 또한 동일하나 각 json object별로 증분 처리함으로써 ObjectId는 고유성을 유지합니다

```
docs = [{'name': 'test1', 'text': 'hello1'}, {'name': 'test2', 'text': 'hello2'}, {'name': 'test3', 'text': 'hello3'}]

collection.insert_many(docs)
```

```
_id: ObjectId('6735da060c97327c32dac0a5')
name : "test1"
text : "hello1"
```

```
_id: ObjectId('6735da060c97327c32dac0a6')
name : "test2"
text : "hello2"
```

```
_id: ObjectId('6735da060c97327c32dac0a7')
name : "test3"
text : "hello3"
```

Python으로 MongoDB collection 읽기

- collection 전체의 내용을 읽을 수도 있습니다

```
names = collection.find()
for name in names:
    print("Found Document:", name)
```

- 조건에 매칭되는 하나의 document 또는 모든 document를 읽을 수도 있습니다

```
name = collection.find_one({"name": "test"})
print("Found Document:", name)
```

```
names = collection.find({"name": "test"})
for name in names:
    print("Found Document:", name)
```

- 업데이트, 삭제동작도 이와 비슷하게 수행할 수 있습니다
- 온라인 강의실의 실습파일을 이용해 각 코드 셀을 실행하고 그 결과를 MongoDB Compass에서 확인해봅시다

중간 정리

- MongoDB에 접속해서 collection와 document를 생성해보았습니다
- MongoDB document의 primary key와 같은 object id의 구조를 알아보았습니다

고가용성의 정의

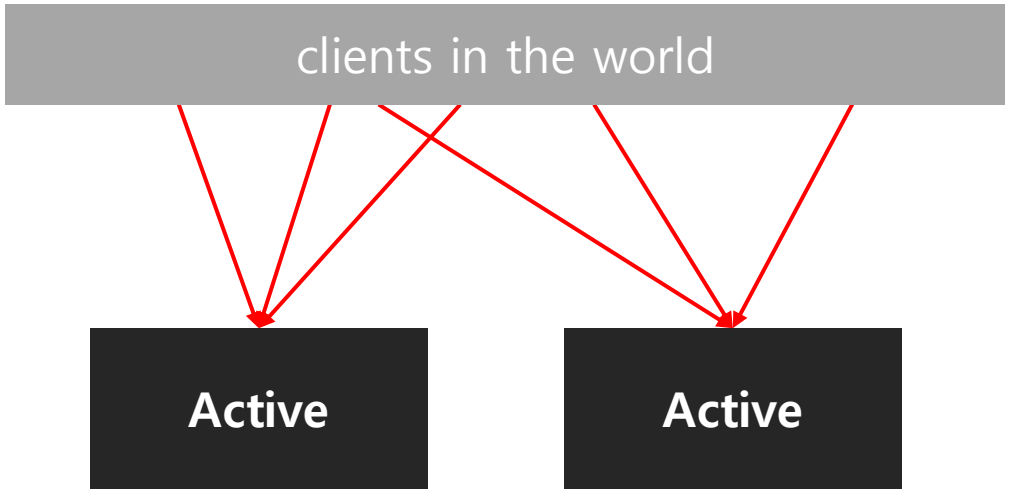
- 서버가 구동중인 컴퓨터는 어떠한 이유로든 중단이 발생할 수 있습니다
- 일부 서버가 중단 되어도 서비스를 지속할 수 있는 수준을 고가용성이라고 합니다
- 고가용성은 로드밸런서나 애플리케이션 아키텍처에 의해 구현됩니다

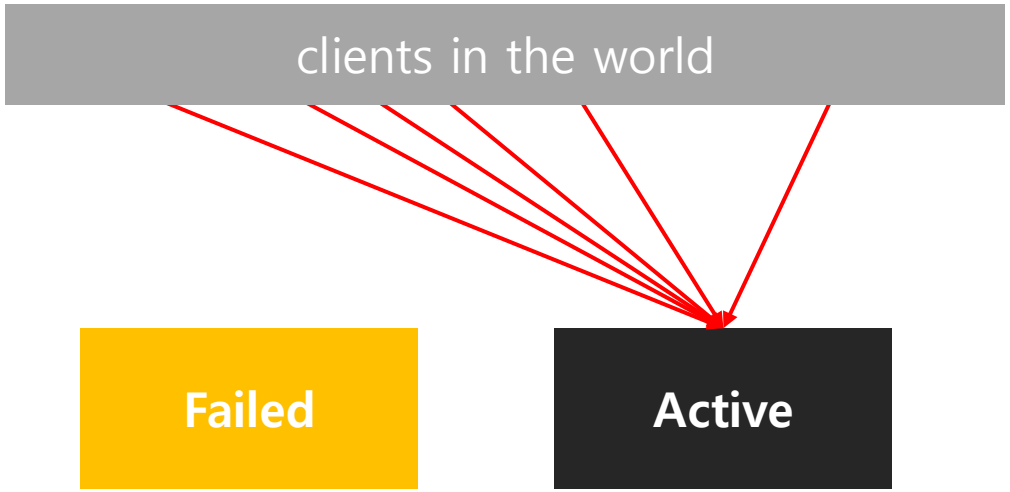
양쪽 모두 서비스

한쪽 서버만 서비스에 사용되고
나머지 한쪽은 동기화만 진행

Active-Active

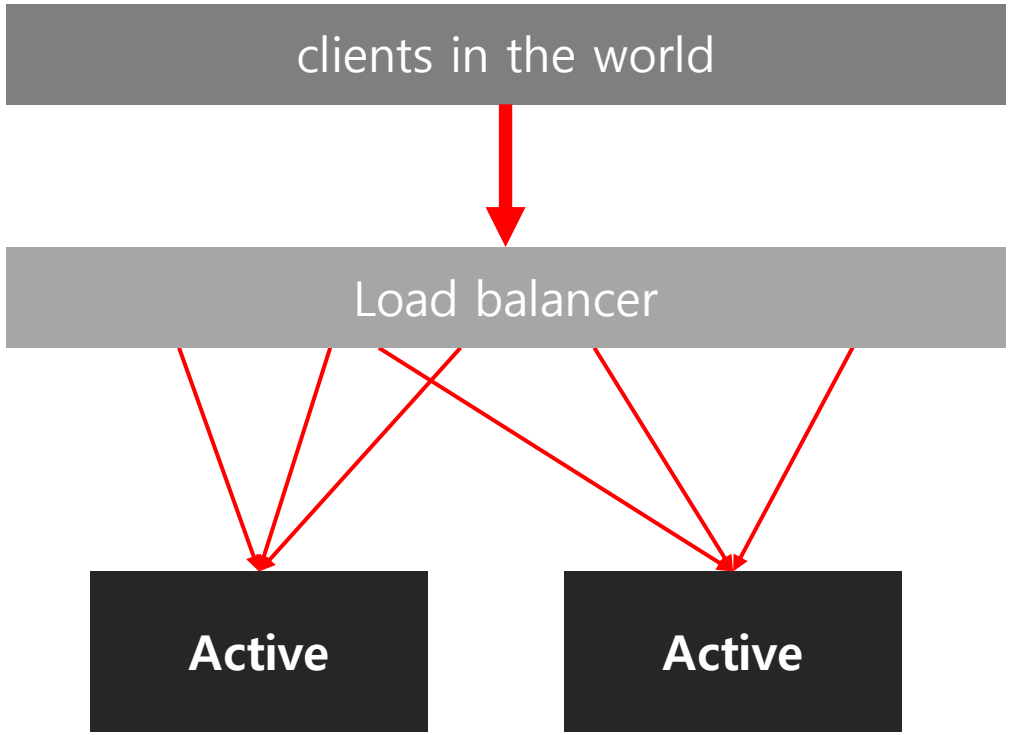
- 2개, 또는 그 이상의 노드가 서비스에 참여합니다
- 만약 1대의 장애가 발생해도 나머지 노드들이 서비스를 이어갈 수 있습니다
- 이 때 클라이언트는 여러대의 서버 중에서 어느 노드를 선택해야 할까요?





로드밸런서

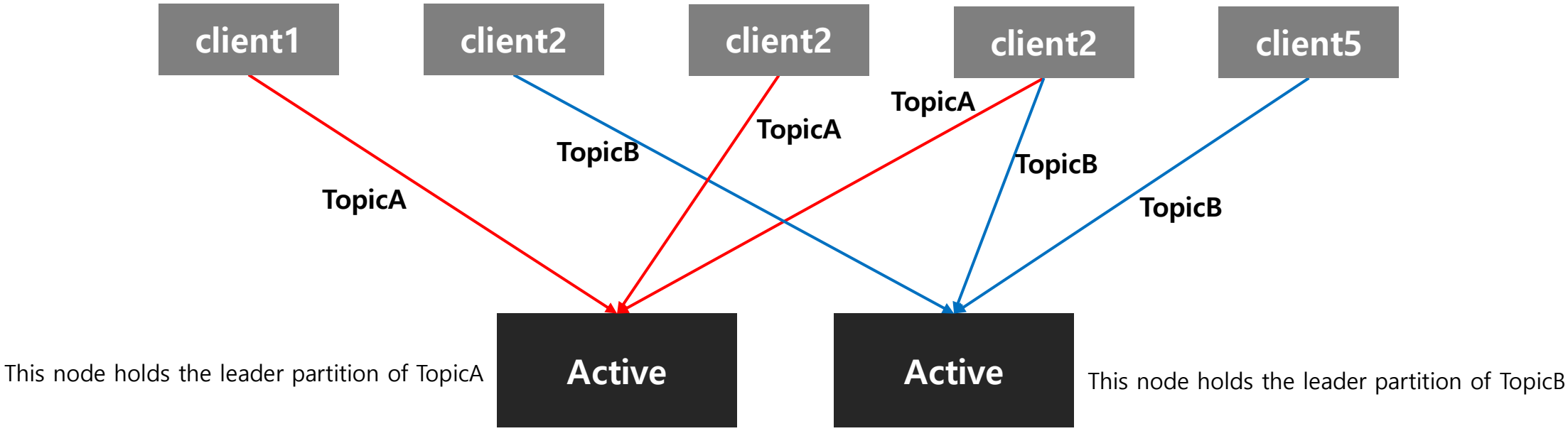
- 네트워크 로드밸런서가 서버들의 전방에 위치하는 경우를 생각해볼 수 있습니다
- 이 경우 클라이언트는 어떤 서버를 골라서 접속할지 고려할 필요가 없습니다



네트워크 로드밸런서는 어플라이언스 형태의 장비
또는 서버내의 소프트웨어로 구현될 수 있습니다

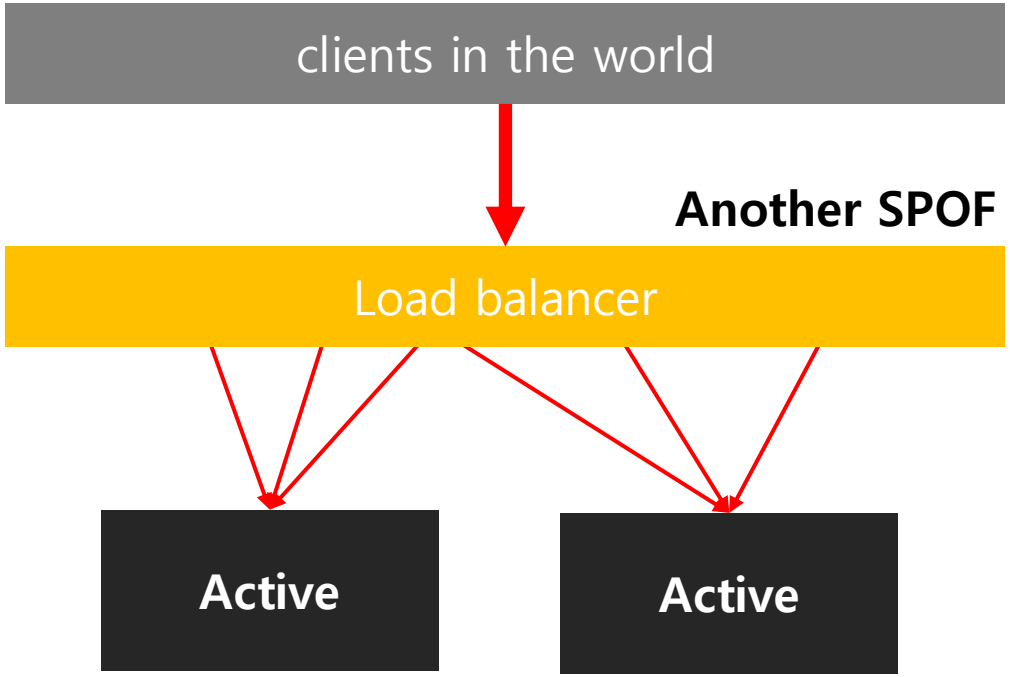
애플리케이션 자체의 매커니즘으로 구현되는 로드밸런서

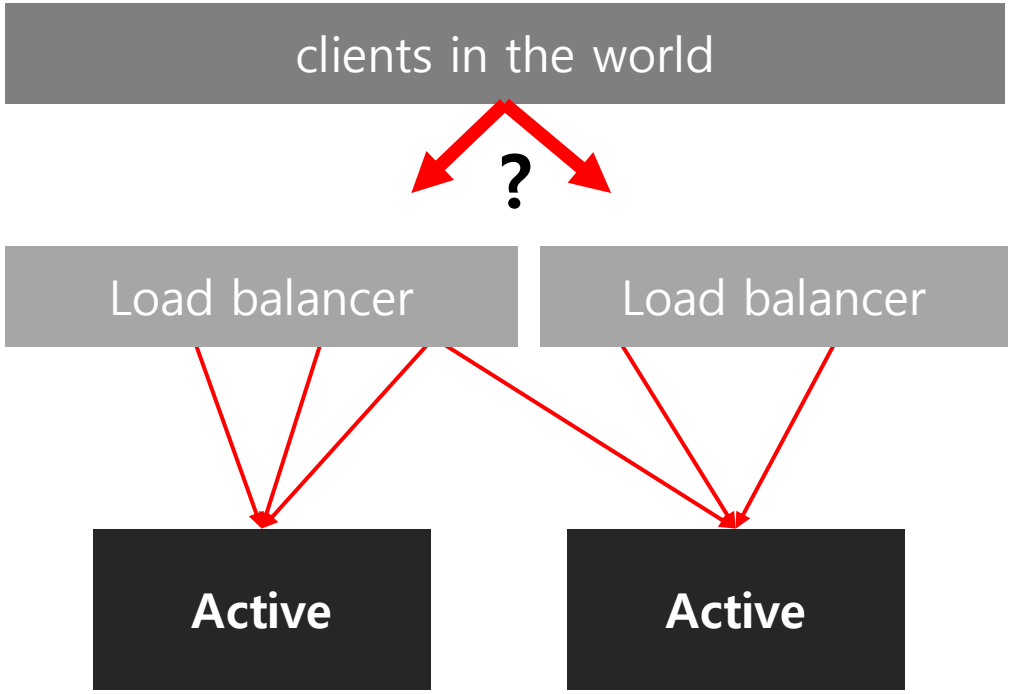
- 클라이언트-서버 프로토콜로 어떤 노드가 선택될지 결정됩니다
- 예를 들면 Kafka의 파티션을 선택하는 것과 같습니다
- 2개의 Active Kafka Broker가 있으며 어떤 Broker의 파티션을 선택할지는 Kafka의 매커니즘에 의해 결정됩니다



로드밸런서와 단일실패점

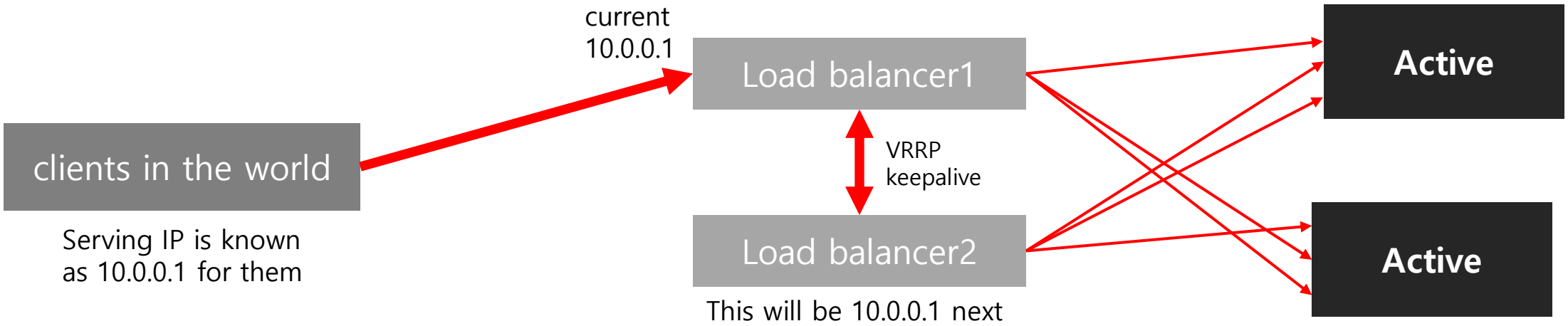
- 로드밸런서에 의해 Active-Active를 구현할 경우 로드밸런서 자체의 장애가 단일실패점이 될 수 있습니다
- 이러한 경우를 예방하기 위해 로드밸런서 자체를 이중화하는 경우도 있습니다
- 그러면 클라이언트는 이제 다시 어떤 로드밸런서를 선택해야 할까요?





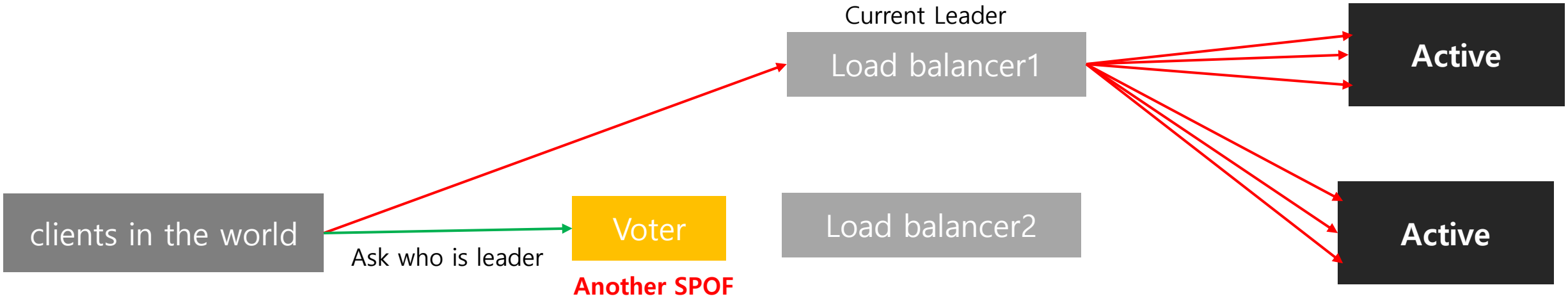
로드밸런서 자체를 Active-Standby로 이중화

- VRRP와 같은 프로토콜로 로드밸런서가 이중화 될 수 있습니다
- VRRP는 TCP/IP와는 다른 프로토콜 입니다
- 하나의 가상 IP를 VRRP그룹중 누가 가져갈 것인지 결정합니다
- 그리고 클라이언트에게는 그 가상의 IP를 공개합니다
- 가상의 IP가 둘 중 누구인지는 중요하지 않습니다
- 그것은 IP 라우팅 레벨에서 결정될 것입니다



애플리케이션 매커니즘에 의한 Active-Standby

- Kafka 파티션의 복제본이 있고 그 중에 Leader를 선택하는 방법과 비슷합니다
- 따라서 Leader선출을 위한 Voter가 필요합니다
- ***Voter가 하나 뿐인 경우 어떻게 될까요 -> 또다른 SPOF***
- 그래서 SPOF를 제거하려면 Voter를 3개나 그 이상으로 구성하게 됩니다

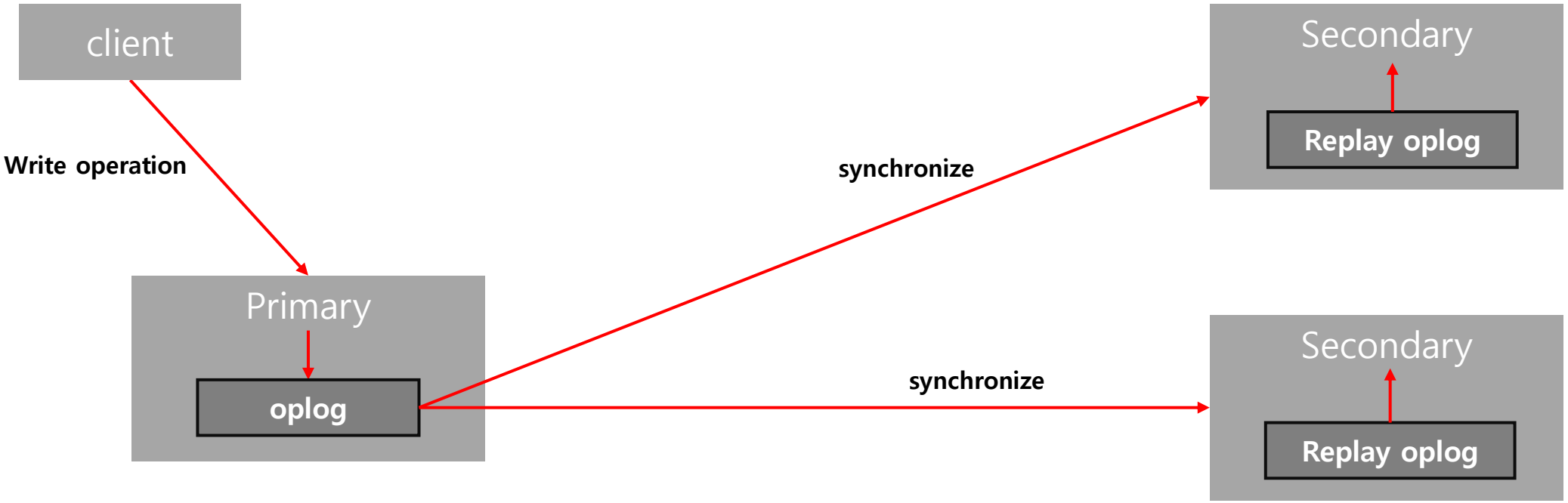


Primary/Secondary node in MongoDB

- MongoDB의 클러스터는 대개 아래와 같이 기본 구성됩니다
- 1 Primary and 2 Secondary
- 쓰기 동작은 Primary 노드에서만 가능합니다
- 읽기 동작은 Secondary 노드에서도 가능합니다
- ***Secondary 노드에서 읽기 수행시 Up-to-date data가 발생할 수 있습니다***

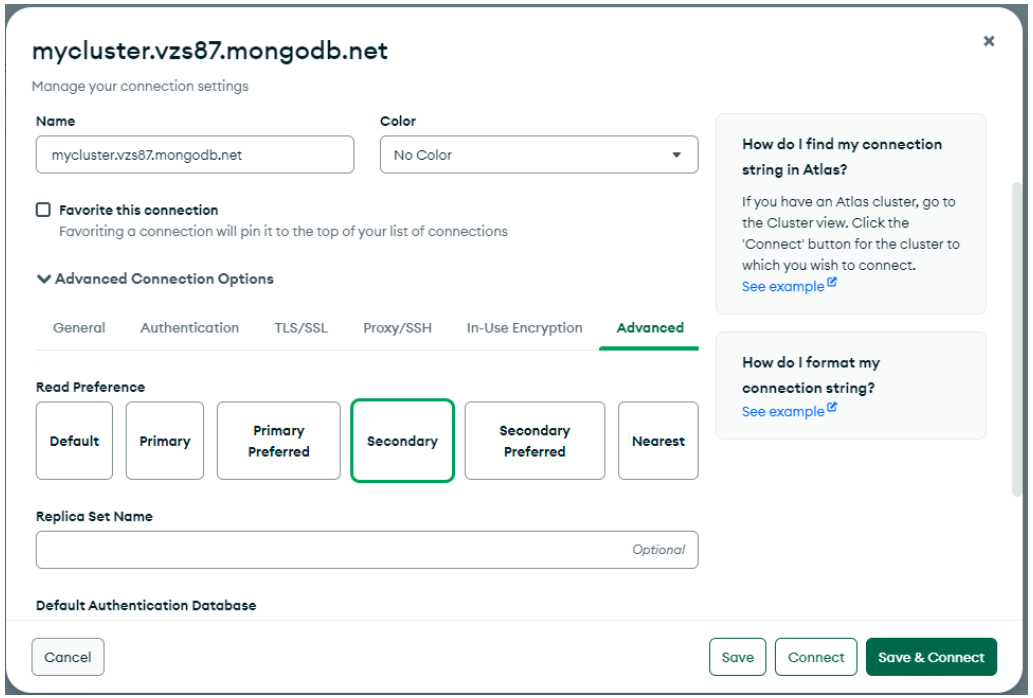
Oplog의 재생에 의한 Primary -> Secondary 복제

- Primary노드는 쓰기동작 후 oplog를 남깁니다
- Secondary노드는 이를 동기화하고 재생하여 복제합니다



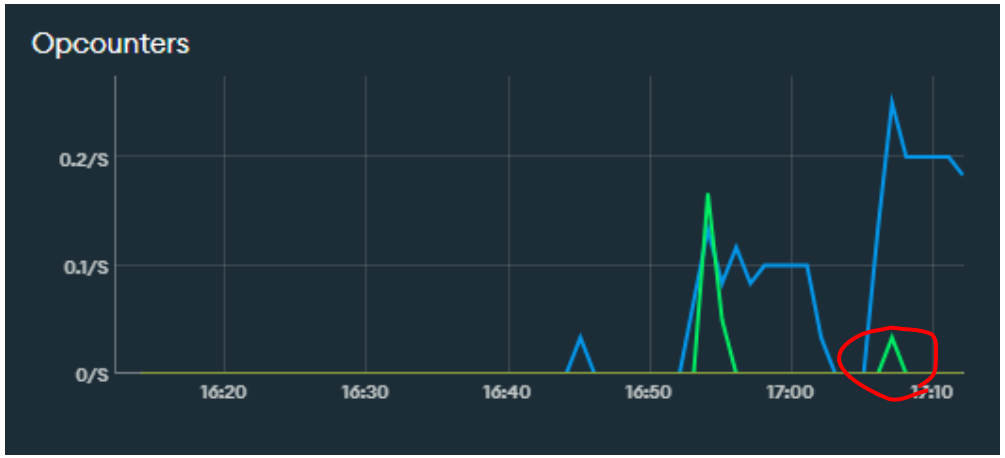
Secondary복제본으로부터 데이터 읽기

- 아래와 같이 Connection string상에 Read Preference설정을 하면 됩니다
- 장점: 읽기 부하 분산
- 단점: up-to-date data가 발생 가능성

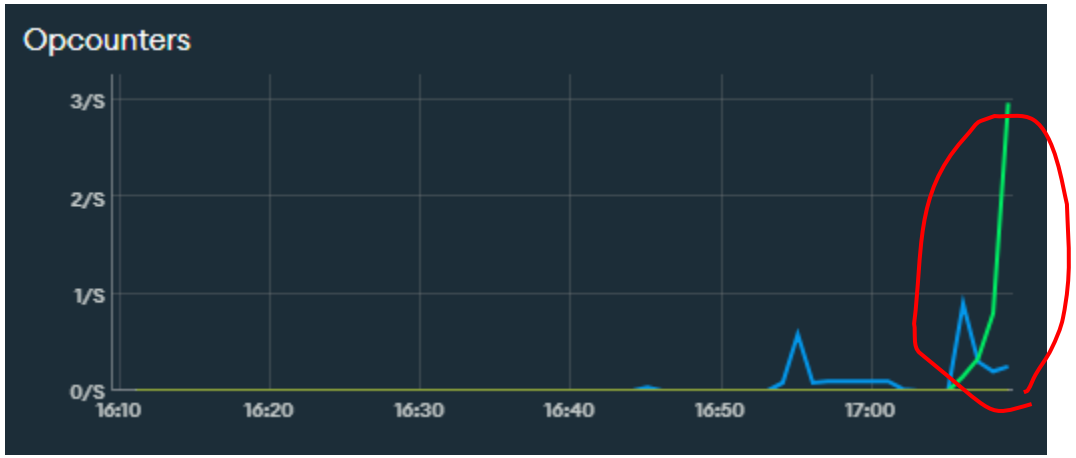


참고 화면

- Read Preference가 Secondary인 경우 Secondary의 Opcounter 증가



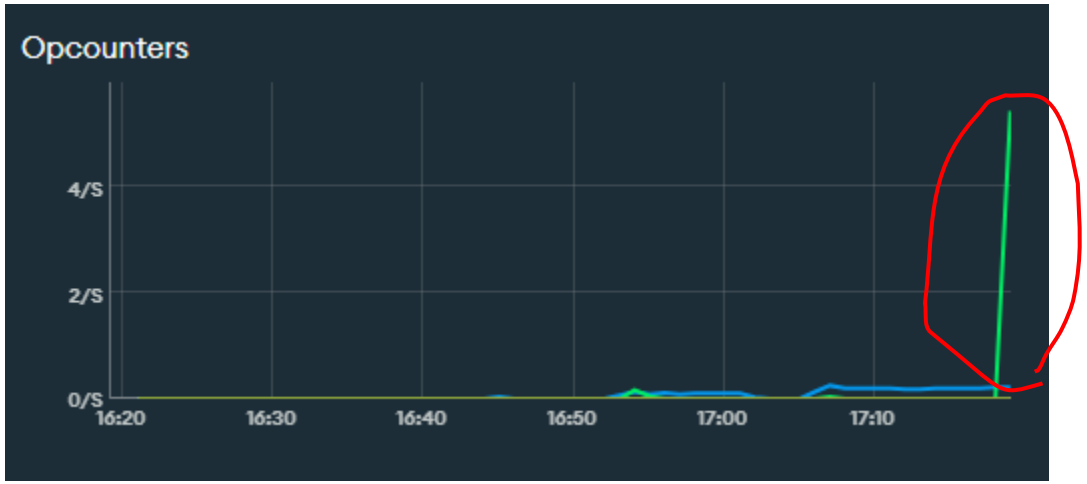
Opcounters of Primary node



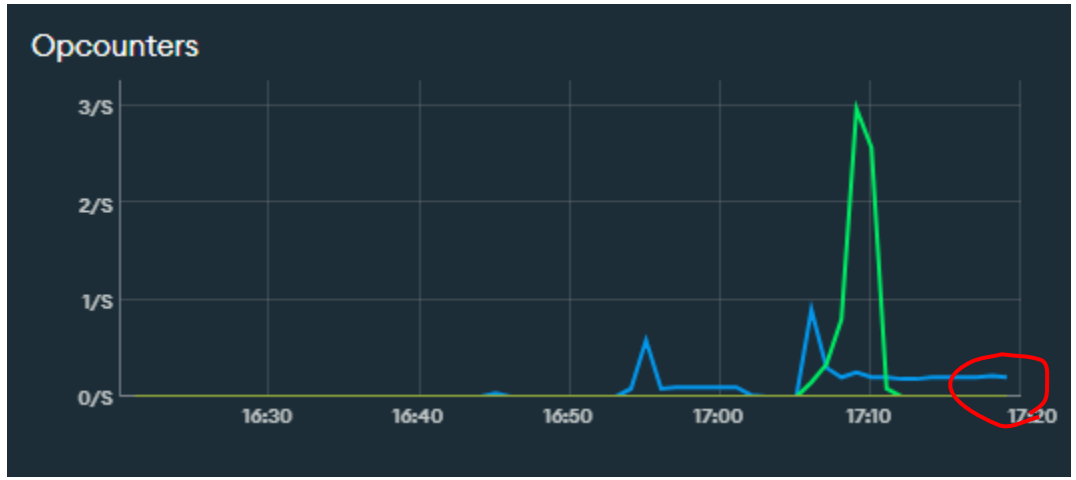
Opcounters of Secondary node

참고 화면

- Read Preference가 Primary인 경우 Primary의 Opcounter 증가



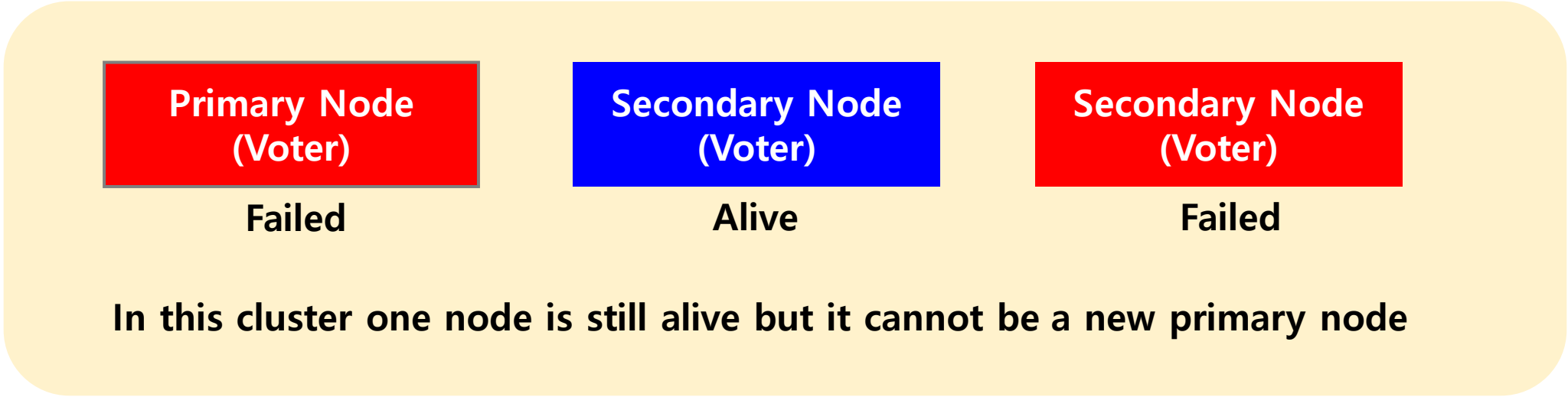
Opcounters of Primary node



Opcounters of Secondary node

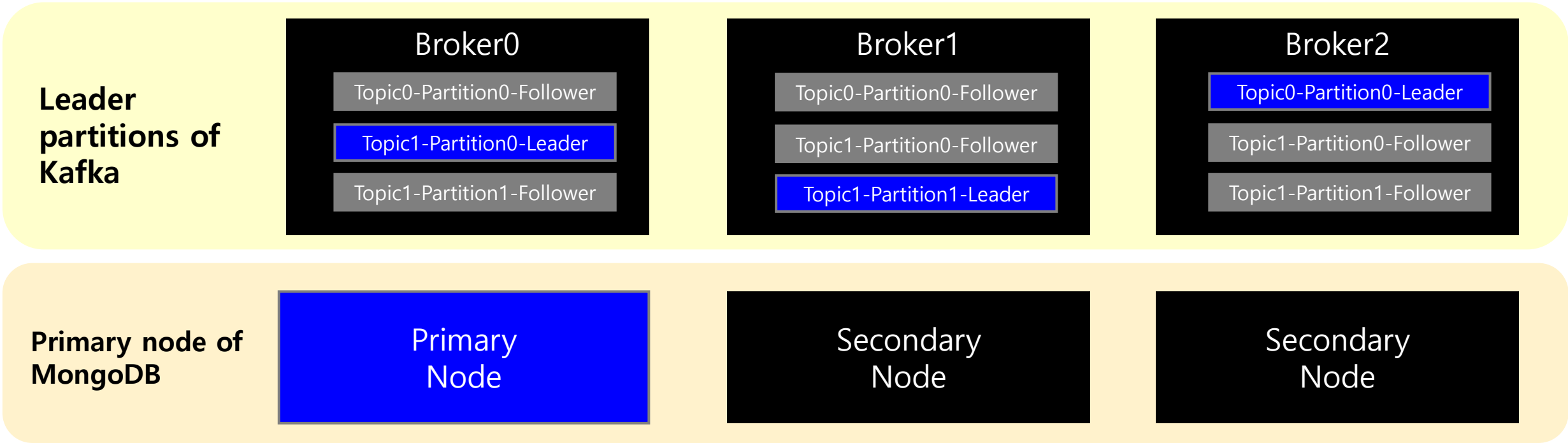
Primary 노드(Leader)의 선출

- Majority election 방식을 따릅니다
- ***이를 위해서는 전체 클러스터 수에서 과반의 노드가 생존하고 있어야 합니다***
- 아래의 경우는 1대의 노드가 생존해 있지만 Majority election 에 실패하여 Primary노드가 되지 못하는 상태입니다



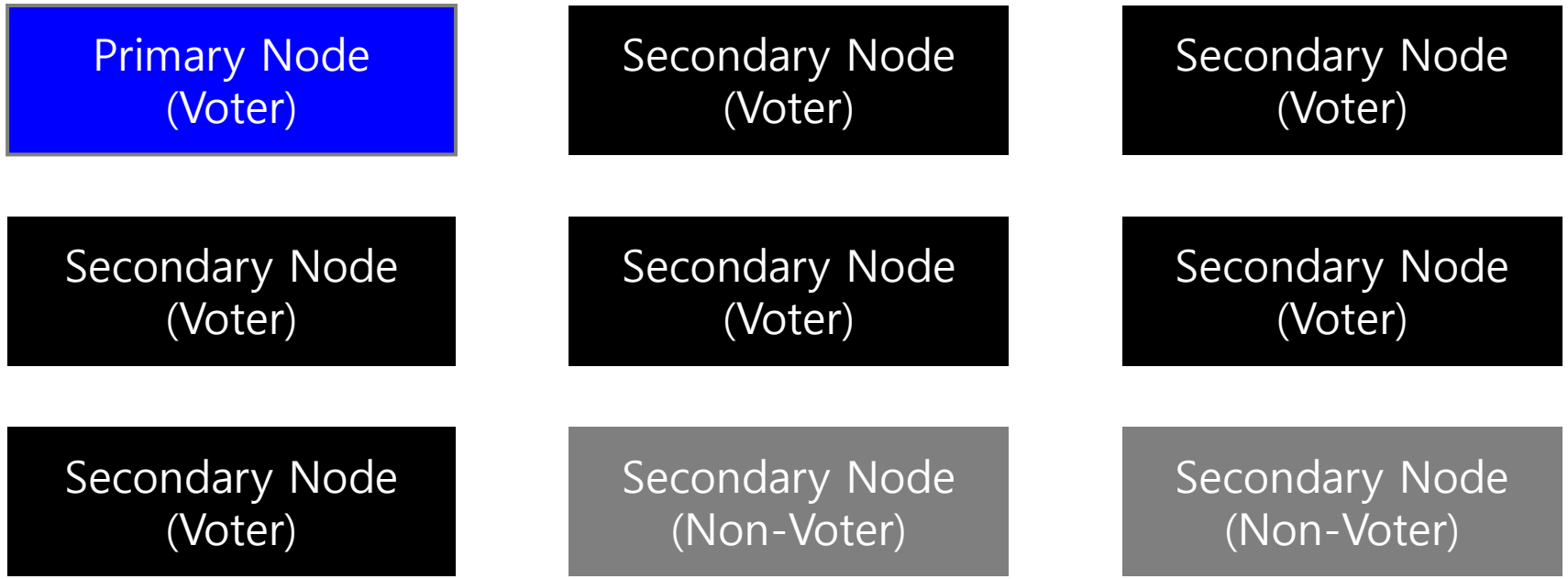
MongoDB와 Kafka의 Leader 선출방식 차이

- Kafka: 파티션에 대한 Leader를 선출
- MongoDB: 노드에 대한 Leader를 선출



Voter가 될 수 있는 노드

- https://www.mongodb.com/ko-kr/docs/manual/core/replica-set-elections/
- Primary노드는 1대이지만 Secondary노드는 여러대를 구성할 수 있습니다
- 이 중에서 누가 Voter역할을 할 수 있을까요? (최대 7대)



MongoDB클러스터에서 Voter역할 설정

- Atlas mongo와 같은 관리형 서비스를 사용할 때에는 이 설정을 하지 않아도 됩니다
- 구축형으로 직접 설치할 때에는 이를 참고하면 됩니다
- https://www.mongodb.com/docs/manual/reference/replica-configuration/
- Majority vote가 성립되기 위해 과반의 노드에 votes설정을 해주면 됩니다
- Majority vote를 통해 선출을 하면 되지만 vote의 효율이 다시 중요해집니다
- 이 문서를 따르면 https://www.mongodb.com/docs/manual/core/replica-set-elections "Member Priority" 라는 것이 존재합니다
- 따라서 Secondary 노드들 중에 priority가 가장 높은 노드가 다음 Primary 노드가 됩니다

```
members: [
  {
    _id: <int>,
    host: <string>,
    arbiterOnly: <boolean>,
    buildIndexes: <boolean>,
    hidden: <boolean>,
    priority: <number>,
    tags: <document>,
    secondaryDelaySecs: <int>,
    votes: <number>
  },
```

If 1 then act as a voter

Kafka의 follower partition으로부터 데이터 읽기

- 실제로 이렇게 동작하지 않으나 가정을 해보는 것입니다
- 만약 이것이 가능하다면 MongoDB처럼 up-to-date data가 발생할 수 있습니다
- 이것이 Kafka에서 Leader파티션만이 읽기/쓰기 활성화되는 이유입니다
- 특히 Kafka는 데이터 파이프라인상에서 전방에 위치하기 때문입니다

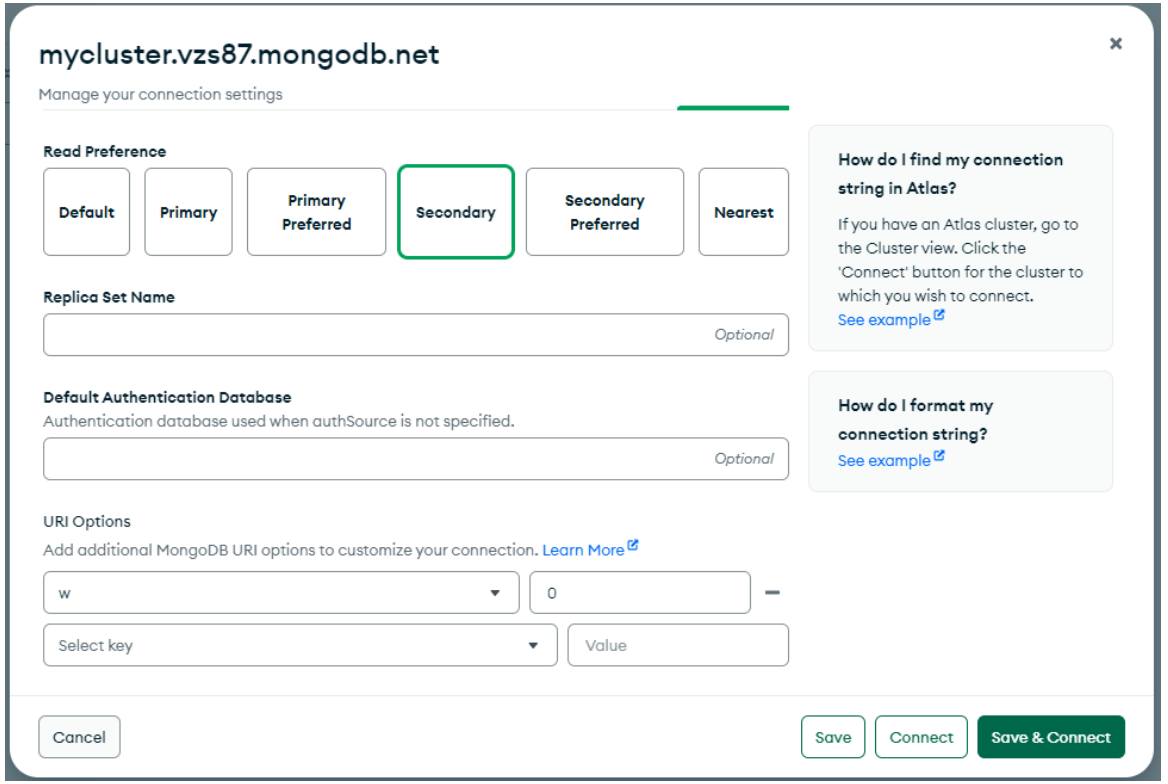
Write Concern

- 쓰기 동작시 Secondary 노드에 동기화를 모두 완료 후 Acknowledgement를 수행해야 할까요?
- 이 수준을 결정하는 값입니다

value of w	description
0	No acknowledgement
1	For primary
greater than 1	For primary and others (totally w nodes)
majority(default)	For primary and others (totally half + 1 nodes)

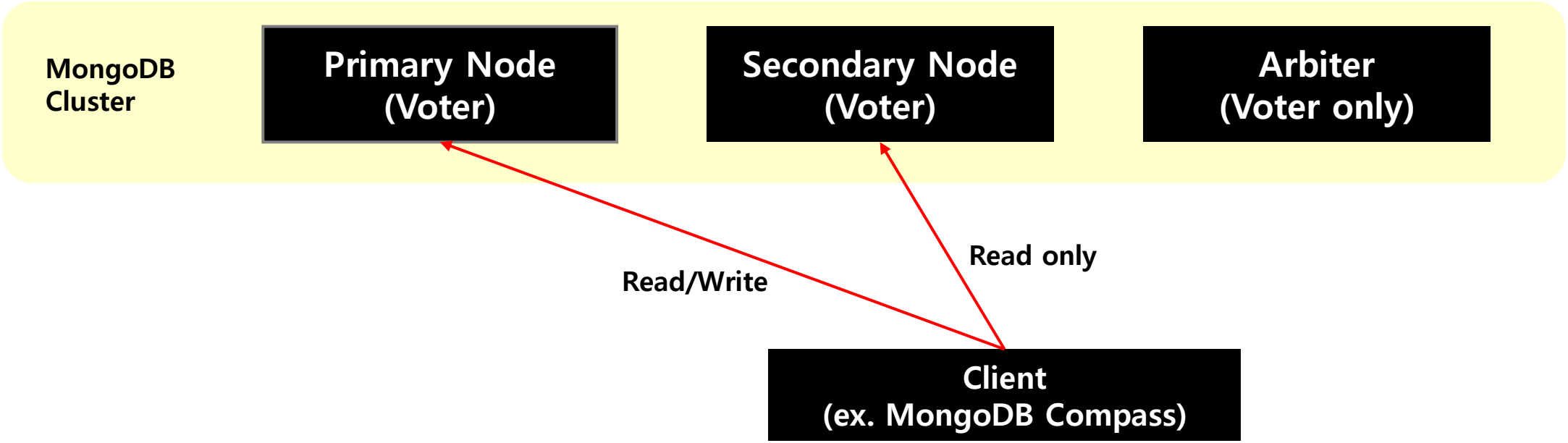
Write concern level의 설정

- MongoDB Compass에서 아래와 같은 설정을 확인해봅니다
- Edit connection > Edit connection string > Advanced > URI Options > choose w



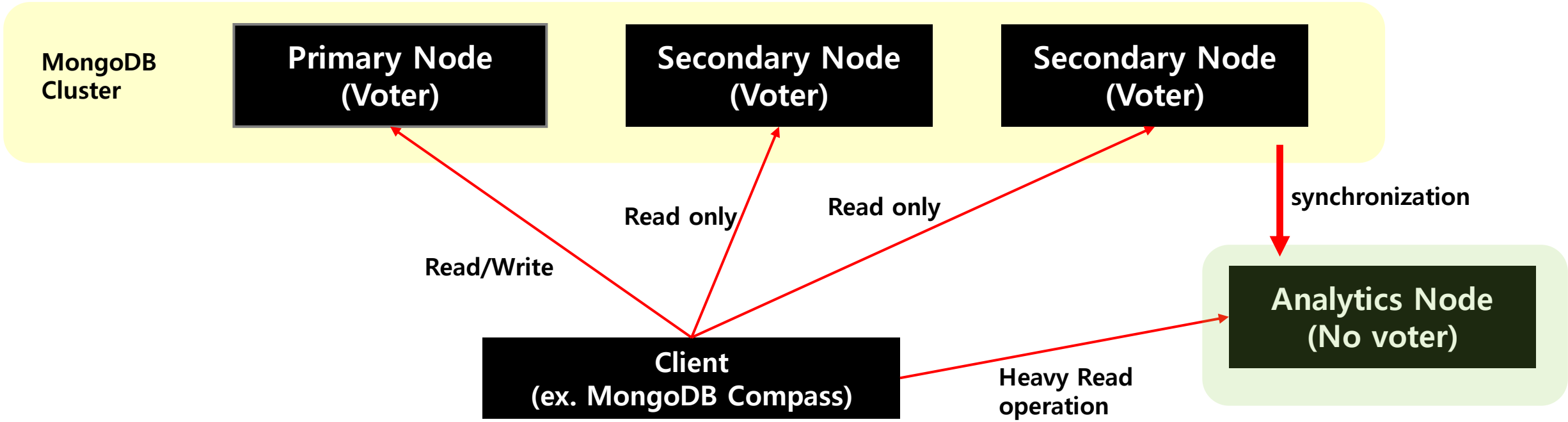
Arbiter node in MongoDB

- 복제본 동기화는 하지 않고 Voter로서만 참여하는 노드입니다
- 복제본 구축 비용을 줄일 수 있습니다



Analytics node in MongoDB

- 워크로드가 독립된 노드입니다
- 데이터 분석등으로 인해 워크로드가 큰 작업을 수행하는 경우 이러한 구성도 가능합니다



정리

- MongoDB의 shard와 replica 에 대해 알아보았습니다
- cluster 형태를 가지는 데이터 관리 시스템이 유지되는 원리가 비슷하다는 것을 확인하였습니다